,

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Empirický výzkum ověřování funkčnosti systému s ohledem na časovou efektivitu za použití manuálních a automatizovaných testů

Empirical research verifying the functionality of the system with respect to time efficiency using automated tests

2017 Michal Krajňák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Michal Krajňák**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Empirický výzkum ověřování funkčnosti systému s ohledem na časovou efektivitu za použití manuálních a automatizovaných testů
Empirical Research Verifying the Functionality of the System with Respect to Time Efficiency Using Automated Tests

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvoření automatizovaných testů za použití testovacích nástrojů nad krabicovým řešením e-shopu FastCentrik od společnosti NetDirect s.r.o a srovnání časové náročnosti nutné pro přípravu Release verze aplikace při použití manuálních a automatizovaných testů. Práce se zabývá empirickým výzkumem ověřování správné funkčnosti s ohledem na časovou efektivitu a validitu za použití manuálních a automatizovaných testů.

1. Představení společnosti NetDirect s.r.o., e-shopové platformy FastCentrik a popis aktuálního procesu vývoje ve společnosti.
2. Popis obecných principů testování, typy testů a jejich použití.
3. Přehled doporučených testovacích nástrojů, popis, porovnání funkcí, srovnání cen, vhodnost použití pro automatizované testování platformy FastCentrik.
4. Vytvoření testovacích scénářů pro Frontendovou část platformy FastCentrik (minimálně 20).
5. Vytvoření testovacích scénářů pro Backendovou část platformy FastCentrik (minimálně 20).
6. Rozdělení aplikace na části podle kritičnosti z pohledu nakupujícího (Frontend) a správce e-shopu (Backend).
7. Mapa pokrytí automatizovaných testů k celkové funkčnosti aplikace.
8. Zapojení testovacích scénářů do procesu Buildu v prostředí Team Foundation Server a Visual Studio 2015.
9. Doporučení pro psaní programového kódu pro bezproblémovou přípravu automatizovaných testů.
10. Porovnání doby trvání testování aplikace při použití manuálních a automatizovaných testů.
11. Celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Michal Rogozný**

Konzultant diplomové práce: Ing. David Ježek, Ph.D.

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Čestné prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 28. apríl 2017

.....

Prehlásenie zástupcu spolupracujúcej právnickej alebo fyzickej osoby

Súhlasím so zverejnením tejto diplomovej práce podľa požiadavku čl. 26, odst. 9 Študijného a skúšobného rádu pre štúdium v magisterských programoch VŠB-TU Ostrava.

V Ostrave 30. marca 2017 .....

Poďakovanie

Rád by som poďakoval vedúcemu diplomovej práce pánovi Ing. Davidovi Ježekovi, Ph.D. a konzultantovi pánovi Ing. Michalovi Rogoznému zo spoločnosti Netdirect, s.r.o. za cenné rady pri práci.

Abstrakt

Cieľom diplomovej práce je vykonanie empirického výskumu overovania funkčnosti systému s ohľadom na časovú efektivitu za použitia manuálnych a automatizovaných testov. V rámci tohto výskumu boli vytvorené automatizované testy pre systém FastCentrik od spoločnosti NetDirect, s.r.o. Tieto vytvorené automatizované testy sú ďalej porovnávané s manuálnym testami v rámci celého životného cyklu. Súčasťou práce je tiež vytvorenie mapy pokrytia automatizovaných testov, rozdelenie systému podľa kritickosti a odporučenia pre písanie programového kódu pre bezproblémové automatizované testovanie.

Kľúčové slová: Telerik Test Studio, Visual Studio, C#, .NET, automatizované testovanie, manuálne testovanie, porovnanie, čas

Abstract

The aim of the master thesis is to make an empirical research verifying the functionality of the system with respect to time efficiency using manual and automated tests. As a part of the research have been developed automated tests for the system FastCentrik from the company NetDirect, s.r.o. These created automated tests are further compared with manual tests throughout the entire life cycle. Part of the master thesis is also to create a coverage map of automated tests, division of the system according to criticality and recommendations for writing programming code for troublefree automated testing.

Key Words: Telerik Test Studio, Visual Studio, C#, .NET, automated testing, manual testing, comparison, time

Obsah

Zoznam použitých skratiek a symbolov

HTML	–	Hyper Text Markup Language
IDE	–	Integrated Development Environment
GUI	–	Graphical User Interface
B2C	–	Business-to-consumer
B2B	–	bussiness-to-business
WPF	–	Windows Presentation Foundation
MVC	–	Model-view-controller
AJAX	–	Asynchronous JavaScript and XML
CSV	–	Comma-separated values
SQL	–	Structured Query Language
XML	–	Extensible Markup Language
URL	–	Uniform Resource Locator
API	–	Application programming interface
DOM	–	Document Object Model
DLL	–	Dynamic Link Library
TFS	–	Team Foundation Server
SaaS	–	Software as a service
SEO	–	Search Engine Optimization

Zoznam obrázkov

Zoznam tabuliek

1 Úvod

Táto diplomová práca sa zaoberá empirickým výskumom overovania funkčnosti krabicového riešenia systému FastCentrik od spoločnosti Netdirect, s.r.o. s ohľadom na časovú efektivitu a validitu za použitia automatizovaných a manuálnych testov. V spoločnosti Netdirect, s.r.o. sú doteraz vykonávané len manuálne testy, kde tester vykonáva testovanie ručne podľa testovacieho plánu, testovacích prípadov a scenárov. Hlavnou nevýhodou tohto typu testovania je jednoznačne doba vykonávania testov a nízka efektivita. Tiež môže byť manuálne testovanie náchylne na chyby, ktoré môžu pri rutinnom a stále sa opakujúcom vykonávaní testov vzniknúť. Tester môže, napr. na niečo pri testovaní zabudnúť alebo je len jednoducho nepozorný. Aby sme dokázali zabrániť takýmto typom chýb, tak je výhodné na takto používané manuálne testy vytvoriť automatizované testy, ktoré umožňujú tento proces testovania zautomatizovať, a tak pomôcť dosiahnuť lepšiu kvalitu softvéru, čím dochádza ku zrýchleniu celého životného cyklu.

Na začiatku práce je popísaná spoločnosť Netdirect, s.r.o., ich e-shopový krabicový systém FastCentrik a aktuálny proces vývoja v spoločnosti. V ďalšej kapitole je popísané testovanie z obecnej perspektívy, rôzne typy testovania a ich použitie. Následne sú popísané doporučené typy testovacích nástrojov, porovnanie cien a vhodnosť ich použitia pre systém FastCentrik. V rámci diplomovej práce vzniknú automatizované testy pre systém FastCentrik, ktoré majú nahradiť spomínané manuálne testy. Tieto testy budú pokrývať ako užívateľskú, tak aj administračnú časť systému. Testy sú zamerané predovšetkým na kritické časti aplikácie, ale aj na menej kritické časti, ktoré sú tiež často využívané zákazníkmi. Vytvorenie testov prebiehalo väčšinou formou implementácie, než s použitím GUI nástroja. Implementácia automatizovaných testov totiž ponúka rozsiahlejšie možnosti pri tvorbe a taktiež dokáže vyriešiť náročnejšie typy testov, ktoré vyžadujú viac zložitosti.

Ďalej práca obsahuje rozdelenie systému na časti podľa kritickosti z pohľadu nakupujúceho a správcu systému. Následne je znázornené pokrytie vytvorených automatizovaných testov vo forme mapy k celkovej funkčnosti aplikácie. V rámci vytvorených testov sú následne popísané odporučenia písania programového kódu pre bezproblémové automatizované testovanie. Tieto odporučenia zahrňujú rady a riešenia, ktoré pomáhajú predísť problémom, ktoré môžu vzniknúť všeobecne pri automatizovanom testovaní, ako aj konkrétne pri danom testovacom nástroji.

V závere práce je popísané porovnanie doby trvania testovanej aplikácie s použitím vytvorených automatizovaných testov oproti manuálnym testom. Toto porovnanie v sebe nezahŕňa len i čas z hľadiska spustenia testov, ale aj celkový čas od návrhu, vývoja, údržby až po nasadenie testov.

2 Spoločnosť NetDirect, s.r.o.

Spoločnosť NetDirect, s.r.o. je softvérová spoločnosť, ktorá sa zaoberá vývojom internetových obchodov a webových stránok. Spoločnosť vznikla v roku 2002 v Ostrave a v súčasnej dobe patrí medzi najvýznamnejších dodávateľov e-business aplikácií na českom a slovenskom trhu. Svoje aplikácie majú postavené prevažne na technológiách od firmy Microsoft a taktiež je spoločnosť držiteľom certifikátu Microsoft Gold Certified Partner. NetDirect stavia riešenia na vlastných produktoch[10]:

- ShopCentrik - je to profesionálny internetový obchod na mieru typu B2C, B2B s napojením na ekonomické, skladové a logistické systémy.
- FastCentrik - veľký e-shop určený pre segment menších firiem a živnostníkov. Je to krabicové riešenie.
- MediaCentrik - redakčný systém pre výstavbu webových firemných prezentácií, portálov a intranetov.

2.1 Systém FastCentrik

FastCentrik je komplexný systém pre obchodovanie na internete. Ide o krabicové riešenie, čo znamená, že systém je dlhodobo vyvíjaný spoločnosťou a tým ponúka pokročilé možnosti nastavenia a prispôsobenia. Tento systém využíva trojvrstvovú architektúru, ktorá pozostáva z prezentačnej, aplikačnej a dátovej vrstvy.

Systém FastCentrik je možné zakúpiť v dvoch verziách a to vo verzii Basic alebo Plus. Medzi funkcie a vlastností, ktoré systém FastCentrik ponúka sú, napríklad[10]:

- Porovnávače tovaru - je možné sa napojiť na porovnávače typu Heureka, Zboží alebo NejlepšíCeny.
- Komplexné štatistiky - umožňuje zobraziť komplexnú štatistiku v administrácii e-shopu.
- Napojenie na ekonomické systémy - POHODA, Altus Vario, Money S3, K2 a Premier.
- Prepojenie s aplikáciami tretích strán.
- Dizajn - rôzne typy šablón, responzívny dizajn.
- SaaS - poskytnutie softvéru zákazníkom formou služby.

2.2 Scrum

Je to jedna z najpopulárnejších agilných metodík. Agilný vývoj na rozdiel od klasického vývoja umožňuje pružne reagovať na zmeny a uprednostňuje sa otvorená komunikácia v tíme.

Týmto spôsobom dokáže agilný vývoj rýchle a efektívne dodať funkčný produkt zákazníkovi. Vo firme Netdirect, s.r.o. je tento typ agilného vývoja zaužívaný.

Scrum je inkrementálnym a iteratívnym rámcom na riadenie produktového vývoja. Cieľom je dodávať najhodnotnejšie vlastnosti, ktoré zákazníci ocenia a potrebujú čo najskôr. Takto je vývoj zameraný na dodávanú hodnotu a nie na konkrétny plán.[6] Táto metodika nám umožňuje:

- dodať funkčný produkt v krátkych cykloch,
- zaručiť rýchlu spätnú väzbu,
- umožniť neustále zlepšovanie,
- rýchlo sa prispôsobiť náhlym zmenám.

2.2.1 Denný Scrum

Jedna z udalostí, ktoré sa využívajú v Scrum procese je denný Scrum, ktorý funguje tak, že každý deň prebieha denné stretnutie celého tímu v pevne stanovenom čase a v stanovenej miestnosti. Dĺžka stretnutia býva väčšinou 15 - 20 minút.

Toto stretnutie prebieha formou komunikácie, kde sa každý člen tímu vyjadrí a oboznámi tím, čo všetko urobil za minulý deň, čo všetko má na pláne dnes a s akými problémami sa stretol. V prípade, ak člen tímu má s niečím problémy, tak mu môžu ostatní členovia tímu pomôcť. Súčasťou tímu je tiež Scrum master, ktorý často rieši problémy mimo oddelenia.

2.2.2 Sprint

Je to základná jednotka vývoja v scrumu. Je to časovo ohraničený úsek na dobu maximálne jedného mesiaca počas ktorého sa vytvára funkčný produkt pre zákazníka. Sprint má väčšinou stanovenú pevnú dĺžku. Nový sprint začína ihneď po dokončení predošlého. Sprint sa skladá z plánovania sprintu, denných stretnutí, vývojárskej práce, recenzie sprintu a z retrospektívy, kde je oznámený progres pre zainteresované osoby a definujú sa úlohy na zlepšenie pre ďalší sprint. V spoločnosti NetDirect, s.r.o. prebieha v intervale štrnástich dní.

2.2.3 Role v Scrum procese

Aby prebiehala metodika Scrum podľa predpisov, tak sú dôležité 3 základné roly:

- Scrum Master - je zodpovedný za odstránenie prekážok, ktoré bránia tímu v dodaní produktu. Komunikuje s tímom a pomáha im dosiahnuť výsledné ciele. Organizuje stretnutia, kontroluje, či tím dodržuje stanovené procesy a snaží sa o to, aby nebol narušený vývojový proces.
- Product Owner - má na starosti porozumenie produktu, komunikáciu so zákazníkom a stanovuje, čo sa má robiť, a ktoré funkčnosti majú aké priority. Je skôr na obchodnej strane

produktu
a väčšinou času trávi v komunikácií so zákazníkom, aby tým mohol zistiť prioritné potreby zákazníka na produkt.

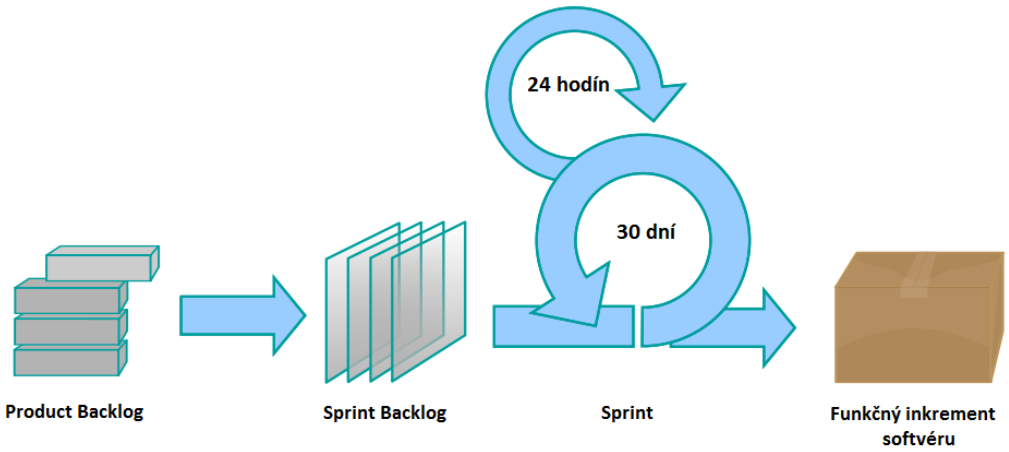
- Vývojový tím - je zodpovedný za dodanie funkčného produktu na konci každého sprintu. Tento tím pozostáva z plnohodnotného tímu vývojárov, analytikov, dizajnérov, testerov atď. Tím by mal byť v rozmedzí 5 - 10 osôb. Pri menej ako 5 osôb môže mať za následok nízku produktivitu a pri viac ako 10 osôb zase môže byť ťažké tím koordinovať.

2.2.4 Scrum proces

Na obr. 1 môžeme vidieť priebeh scrum procesu. Na začiatku procesu sú definované a zhromažďované požiadavky na produkt v produktovom backlogu. Tieto požiadavky má na starosti vlastník produktu, ktorý im priraďuje aj určité priority. Produktový backlog nikdy nie je úplný, a preto je dynamický a môže sa v priebehu meniť.

Následne sú vývojárskym tímom vybrané najprioritnejšie požiadavky z produktového backlogu do sprint backlogu. Sprint backlog je teda súbor týchto vybraných položiek z produktového backlogu. Podstatná časť tímov dokáže určiť, koľko času dokáže zabrať určitá úloha. Prioritné požiadavky sú vyberané tak, aby bol tím schopný reálne vyvinúť, otestovať a integrovať tieto požiadavky v stanovenej dĺžke iterácie. Pokrok jednotlivých požiadaviek v rámci iterácie je diskutovaný na denných stretnutiach.[6]

Na konci každej iterácie resp. sprintu je dodávaný funkčný produkt zákazníkovi a počas toho je možné získať spätnú väzbu od zákazníka. Zároveň je organizovaná porada, kde celý tím zhodnotí svoj výkon. Tento celý kolobeh sa iteračne opakuje až dovtedy, pokým nie sú splnené všetky požiadavky.



Obrázok 1: Scrum proces[6]

3 Obecne o testovaní

Testovanie je proces spustenia programu s cieľom nájsť chyby. Cieľom testovania je overiť, či výsledný systém spĺňa predpísané požiadavky. Overenie by malo byť vykonané v čo najširšom rozsahu, tak aby prípadné nedostatky systému boli odhalené včas a nie až pri využívaní systému užívateľom. Testovanie samo o sebe priamo nezvyšuje ani nezaisťuje kvalitu produktu, poskytuje však pre túto činnosť potrebné vstupy. Je to potenciálne nekonečný proces a o jeho zastavení musí byť rozhodnuté na základe uváženia rôznych faktorov, ako je jeho postupne sa znižujúca efektivita, časové a rozpočtové obmedzenie, vykonanie všetkých kľúčových testov a podobne.[1]

I samotné testy by mali byť otestované, pretože aj testeri sú len ľudia a dokážu urobiť chybu ako ostatní. Vzájomná revízia vytvorených testov medzi testermi je často veľmi efektívna. Testovanie softvéru je teda nevyhnutné aplikovať pre akýkoľvek druh softvéru, pretože my všetci robíme chyby a tieto chyby síce niekedy môžu byť len zanedbateľné, avšak v niektorých prípadoch môžu byť až nebezpečné. Preto netreba testovanie zanedbávať a malo by sa s ním začať už vo fáze špecifikácie požiadaviek, pretože, ak by sa s ním začalo neskôr, tak to môže mať fatálne následky pri nasadení systému k zákazníkovi.

3.1 Dôležité pojmy v testovaní

Aby sme lepšie pochopili ako funguje testovanie softvéru, tak je dôležité byť s nasledujúcimi pojmami oboznámený.

3.1.1 Chyba

Typicky v každom softvéri sa nájde nejaká chyba. I keď zavádzame množstvo testovania a toto testovanie prejde v poriadku, aj tak nám to nezaručí, že v softvéri sa nenachádza chyba.

Chyba je následkom ľudskej činnosti, ktorá vytvára nesprávny výsledok. Táto chyba môže byť spôsobená programátorom, ktorý nesprávne porozumel požiadavkám systému, alebo pri implementácii nesprávne spočítal určité hodnoty.[5]

3.1.2 Porucha

Je základným dôvodom pre zlyhanie softvéru. Porucha môže vzniknúť chybným kódom alebo hardvérom.[5]

3.1.3 Zlyhanie

Zlyhanie nastane, ak sa vykoná chybný kód. Zlyhanie je neschopnosť systému vykonávať požadované funkcie podľa zadaných špecifikácií.[5] To znamená, že ak tester vykoná testovanie v ktorom je chybný kód, tak to vedie k zlyhaniu systému.

3.1.4 Testovací plán

Je to dokument, ktorý popisuje prístup k testovaniu systému. Obsahuje stratégiu testovania, zdroje použité na testovanie, potrebné testovacie prostredie, zoznam testovacích prípadov, zoznam testovaných funkcií, harmonogram testovacích aktivít a podobne.[2]

3.1.5 Testovací prípad

Je to zostava vstupov, podmienok pre spustenie a očakávaných výsledkov pre overenie súladu s konkrétnou požiadavkou.[1] Týmto testovacím prípadom overujeme, či systém reaguje na požadované vstupy tak, ako by mal.

3.1.6 Testovací scenár

Súbor niekoľkých testovacích prípadov tvorí testovací scenár.[2] Testovacie scenáre sú všetky funkcie, ktoré je potrebné otestovať. Typickým príkladom testovacieho scenára by mohla byť kontrola funkčnosti prihlásenia. Táto funkčnosť prihlásenia môže mať viacero testovacích prípadov ako napr. overenie pre platný vstup, pre neplatný vstup atď.

3.2 Typy testov

Každý vývoj softvérového produktu prebieha v jednotlivých fázach a každá fáza si vyžaduje iný typ testovania. V tejto podkapitole budú popísané základné typy testovania.

3.2.1 Manuálne testovanie

Manuálne testovanie je testovací proces, v ktorom tester postupuje podľa určitého testovacieho plánu, testovacieho prípadu a scenára. Pri tomto type testovania vykonáva tester všetko ručne za počítačom. To znamená, že testovanie je vykonávané bez akéhokoľvek testovacieho nástroja.

Tester v podstate preberá úlohu koncového užívateľa a snaží sa identifikovať potenciálne chyby a neočakávané správanie systému. Zapisuje si všetky testovacie scenáre, prípady, aktuálne výsledky, očakávané výsledky a vzniknuté chyby hlási vývojárom.

Výhody manuálneho testovania:

- manuálne testovanie je jednoduché a môže ho vykonávať aj menej kvalifikovaný človek,
- je vhodné použiť na malé aj väčšie projekty,
- oplatí sa použiť pri testovaní grafického rozhrania, kde dochádza k častým zmenám,
- je výhodné aplikovať na testy, ktoré vyžadujú ľudský nadhľad a chápanie.

Nevýhody manuálneho testovania:

- pri väčšom množstve testov je takýto typ testovania nudný a pre testera aj náchylný k chybám, pretože môže, napr. na niečo zabudnúť,
- je časovo náročné manuálne testovať veľké množstvo testov,
- manuálne testovanie je možné vykonávať len vtedy, keď daný tester je k dispozícií, čo pri automatizovanom testovaní nemusíme riešiť.

3.2.2 Automatizované testovanie

Je typ testovania, ktorý nám umožňuje testovať softvér spustením testovacieho skriptu v testovacom nástroji. Tento typ testovania nám umožňuje zautomatizovať manuálny proces testera, a tak docieliť rýchlejšie a efektívnejšie výsledky. Výhody automatizovaného testovania:

- je vhodné automatizované testovanie aplikovať na veľké množstvo testov a tým zabrániť chybám, ktoré vznikajú pri manuálnom testovaní,
- automatizované testovanie je časovo úsporné, pretože dokáže vykonávať testy bez prítomnosti testera,
- väčšina testovacích nástrojov na automatizované testovanie ponúka množstvo výhod, ktoré by sme pri manuálnom testovaní nemali,
- používa sa hlavne pri dlhodobých projektoch, kde sa ich uplatnenie dokáže prejaviť,
- umožňuje spustiť test s veľkým množstvom dát.

Nevýhody automatizovaného testovania:

- je náročný na údržbu, pretože pri nejakej zmene vyžaduje zmenu programového kódu,
- nie je praktický na testovanie grafického rozhrania, pretože tam často dochádza ku zmenám,
- vyžaduje mať kvalifikovanejšieho človeka, ktorý má aspoň základné znalosti v programovaní.

3.2.3 Jednotkové testovanie

Jednotkové testovanie resp. unit testing je testovanie, ktoré prebieha na menších častiach systému, ktorým hovoríme jednotky. Touto menšou časťou sa obvykle myslí nejaká funkcia, procedúra alebo trieda.[4] Každý programátor, ktorý napíše nejaký kód, tak by si ho mal týmto jednotkovým testom otestovať, či funguje správne. Existujú nástroje, ktoré nám umožňujú tento typ testov vykonať. Napríklad pre jazyk C# to môže byť nástroj NUnit a pre jazyk Java nástroj JUnit.

3.2.4 Systémové testovanie

Je to testovacia technika, ktorá využíva spôsob testovania čiernej skrinky a je vykonávaná za účelom vyhodnotenie zhody celého systému so špecifickými požiadavkami. Je to postupnosť rôznych testov, ktoré kontrolujú celý systém ako napr. hardvér, softvér, prostredie, databáza atď. Keďže táto technika spadá do testovania čiernej skrinky, tak by nemala vyžadovať žiadne znalosti o vnútornej štruktúre kódu a logiky.

Existuje množstvo typov systémového testovania. V nasledujúcich deleniach budú uvedené tie najpoužívanejšie.

3.2.4.1 Testovanie použiteľnosti

Je nefunkčná testovacia technika, ktorá je meradlom toho, ako jednoducho môže systém používať koncový užívateľ. Toto meranie je ťažké určiť, ale môžu nám k tomu pomôcť nasledujúce parametre[5]:

- úroveň zručnosti na používanie systému,
- potrebný čas zvyknúť si na používanie systému,
- posúdenie prístupu užívateľov k používaniu softvéru.

3.2.4.2 Výkonnostné testovanie

Je to nefunkčná testovacia technika, ktorá má určiť systémové parametre z hľadiska odozvy a stability pri rôznom zaťažení. Výkonnostné testovanie meria znaky kvality výkonu, ako napr. škálovateľnosť, spoľahlivosť a využitie systémových prostriedkov.[5]

3.2.4.3 Záťažové testovanie

Je to najjednoduchšia testovacia technika výkonnostného testovania pomocou ktorej sa odozva systému meria v rôznych podmienkach zaťaženia. Vykonáva sa za normálnych a vysokých podmienkach zaťaženia.[5] Typickým príkladom záťažového testovania môže byť sťahovanie série veľkých súborov z internetu alebo tiež beh viacerých aplikácií súčasne.

3.2.4.4 Regresné testovanie

Cieľom je otestovať, či odstránením chyby alebo zmenou v aplikácií nevznikli nové chyby. Je to v podstate opakovanie testov s cieľom zaistiť, že nie sú spôsobené žiadne zmeny v softvéri.

3.2.4.5 Test hraničnej záťaže

Je to ďalší typ výkonnostného testovania, ktorý sa používa pri veľkej záťaži na systém a slúži nám na to, aby sme zistili, či nedôjde k neočakávaným chybám, ktoré za normálnych okolností nenastanú.

3.3 Spôsoby testovania

Na testovaný systém sa môžeme pozerať z rôznych perspektív. Existujú 3 typy testovacích metód, ktoré nám pomáhajú pozerať sa na testovací systém inými očami.

3.3.1 Testovanie čiernej skrinky

Metóda testovania čiernej skrinky tzv. black box testing je metóda, ktorá nám umožňuje pozerať sa na systém z pohľadu užívateľa. To znamená, že síce vieme ako ten systém funguje, ale nevieme ako pracuje vo vnútri, nepoznáme presnú štruktúru systému. Tester nemá prístup k zdrojovému kódu a vykonáva len interakciu medzi systémom definovaním vstupov a následným získaním očakávaných výstupov. Tester vychádza len zo špecifikácie požiadaviek systému.[4]

Výhodou takejto metódy testovania je, že tester sa nemusí zaoberať implementáciou a tým väčšie množstvo menej kvalifikovaných testerov dokáže takýto typ testovania vykonať. Taktiež je výhodou, že nie sme závislí na implementácii, to znamená, že môže sa síce zmeniť programovací jazyk, databáza alebo operačný systém, naše testovacie scenáre však ostanú nedotknuteľné.

Na druhej strane tu však vidíme aj nevýhodu v tom, že tester ma značné obmedzenia na testovanie aplikácie a je ťažké pre neho definovať všetky možné vstupy pre systém. Tým, že nepoznáme implementáciu a nemáme prístup k zdrojovému kódu, tak nemôžeme presne vedieť či aplikácia funguje ako má.

3.3.2 Testovanie bielej skrinky

Metóda testovania bielej skrinky tzv. white box testing je metóda, ktorá na rozdiel od testovania čiernej skrinky pozná vnútornú štruktúru systému. Tester má k dispozícií zdrojový kód systému a dokáže lepšie odhaliť chyby, ktoré nie je možné inak odhaliť.[4]

Hlavnou výhodou je, ako sme spomenuli, lepšie odhalenie chýb, ktoré máme pri dostupnosti zdrojového kódu aplikácie. Ďalšou podstatnou výhodou je, že môžeme odhaliť nežiaduci kód v testovanej aplikácie. To znamená, že nejaký programátor možno svojím nedopatrením zabudol odstrániť svoj kód, ktorý využíval, napr. pri ladení aplikácie, a tým môže tento kód v konečnom dôsledku spôsobovať značné problémy. Ak by, napr. išlo o bankovú aplikáciu, tak by sa mohlo stať, že nežiaduci kód by mohol prevádzať finančné prostriedky na iný účet. Takže tu vidíme tiež značnú výhodu testovania bielej skrinky.

Nevýhodou tejto metódy testovania je, že tester musí mať určité znalostí v programovaní, aby mohol porozumieť implementácii testovanej aplikácie, a tak mohol daný systém dobre otestovať. Ďalšiu nevýhodou je, že tento typ testovania je aj dosť nákladný, pretože vyžaduje mať k dispozícii rôzne ladiace nástroje a analyzátory.

3.3.3 Testovanie sivej skrinky

Táto testovacia metóda sa síce veľa nespomína, ale tiež stojí za zmienku. Metóda testovania sivej skrinky tzv. grey box testing je metóda, ktorá je kombináciou metódy testovania čiernej

skrinky a bielej skrinky.[4] Pri tejto metóde síce máme informácie o vnútornej štruktúre systému, ale nie až tak veľa, aby sme to mohli považovať za testovanie bielej skrinky. Táto metóda testovania sa najviac využíva pri webových aplikáciách.

4 Porovnanie testovacích nástrojov

Pre použitie automatizovaného testovania je nutné, aby sme k tomu využili nejaký testovací nástroj. Tento nástroj umožňuje testerovi vytvoriť automatizovaný test formou testovacieho skriptu. Tester má na výber, či chce test vykonať cez GUI nástroj formou nahrávania krokov, alebo cez implementáciu, kde už tester musí mať určité znalosti z programovania. V tejto kapitole budú porovnávané najpoužívanejšie testovacie nástroje.

4.1 Telerik Test Studio

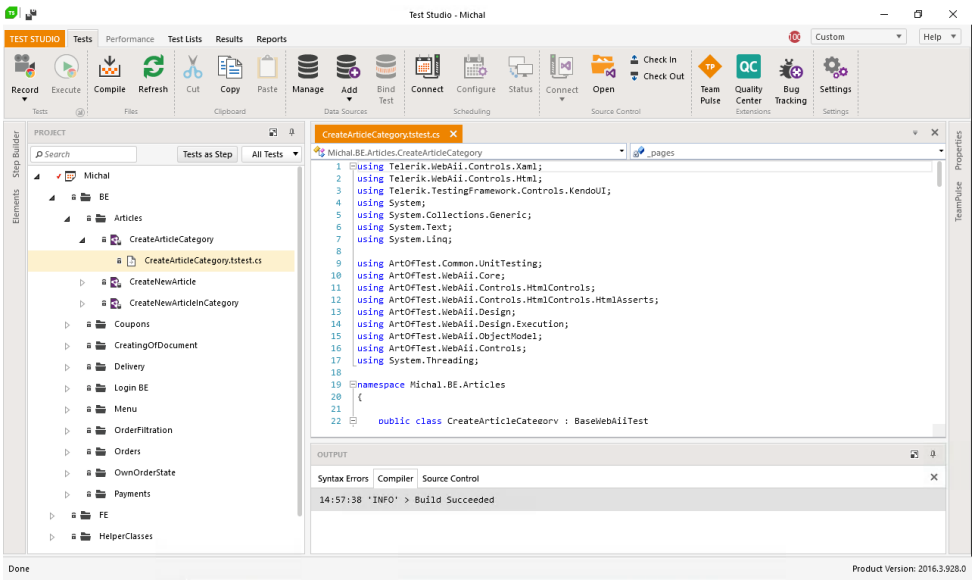
Telerik Test Studio je súbor nástrojov, ktoré nám umožňujú vytvoriť automatizované regresné a funkčné testy pre všetky typy webových aplikácií, Silverlight a Desktop aplikácií napísané v knižnici WPF. Umožňuje tiež vykonať záťažové testovanie, výkonnostné testovanie, funkčné testovanie a testovanie mobilných zariadení pre platformy iOS a Android.[7] Pri tomto nástroji máme dve možnosti ako vykonať automatizovaný test:

- nahrávanie skriptu a následne prehrávanie,
- vytvorením skriptov v jazyku C# alebo VB.NET.

Tento nástroj je ponúkaný spoločnosťou Telerik v dvoch variantoch:

- ako samostatný testovací nástroj s prostredím,
- ako integrácia do vývojového prostredia Microsoft Visual Studio.

Na obr. 2 môžeme vidieť ukážku samostatného testovacieho nástroja Telerik Test Studio.



Obrázok 2: Telerik Test Studio[6]

4.1.1 Výhody

- V Telerik Test Studiu môžeme nahrávať a následne prehrávať testovacie skripty na prehliadačoch Google Chrome, Safari, Firefox a Internet Explorer. Týmto máme zaručenú veľkú podporu webových prehliadačov, čo môže byť veľmi užitočné, keď chceme vedieť ako systém reaguje na rôzne webové prehliadače.

- Umožňuje nám podporovať aplikácie, ktoré sú vyvinuté v technológiách AJAX, HTML5, JavaScript, ASP.NET MVC, Ruby, Silverlight a WPF.

- V Telerik Test Studiu majú testeri možnosť písať aj manuálny skript, čo môže veľmi zefektívniť prácu, pretože okrem zápisu samotného testovacieho prípadu môžu testeri si zaznamenať obraz z aktuálneho kroku.

- Telerik Test Studio tiež podporuje funkciu data-driven testovanie. Data-driven testovanie je pojem, ktorý sa používa v testovaní softvéru a umožňuje nám rovnakú sekvenciu testovacích krokov vykonať opakovane a to s použitím dátového zdroja. Test Studio podporuje až 5 typov dátových zdrojov[7]:
  - lokálny dátový zdroj,
  - excel súbor,
  - XML súbor,
  - CSV súbor,
  - SQL databáza.

- Nástroj umožňuje znovupoužitie testov, čo je výhodne v prípade, ak sa opakujeme pri ďalších testoch a nechceme mať zbytočné duplicitné kódy, tak môžeme jeden test napojiť na druhý.

- 24-hodinová podpora a to dokonca aj pri skúšobnej verzii nástroja.

4.1.2 Nevýhody

- Testeri majú možnosť písať testovací skript v jazyku C# alebo VB.NET, takže nám z toho vyplýva, že sme pri testovaní obmedzení na operačný systém Microsoft Windows.

- Na rozdiel od iných testovacích nástrojov je Telerik Test Studio spoplatnený. Ak chceme automatizované testovanie pre webové a desktopové aplikácie, tak je cena za licenciu 2499 dolárov. Pri tejto cene to však je aj pochopiteľne, pretože obsahuje, na rozdiel od iných nástrojov, množstvo vlastností a funkčností.

- Treba mať výkonnejší počítač, pretože Telerik Test Studio má veľa funkčností, ktoré vyžadujú výkon.

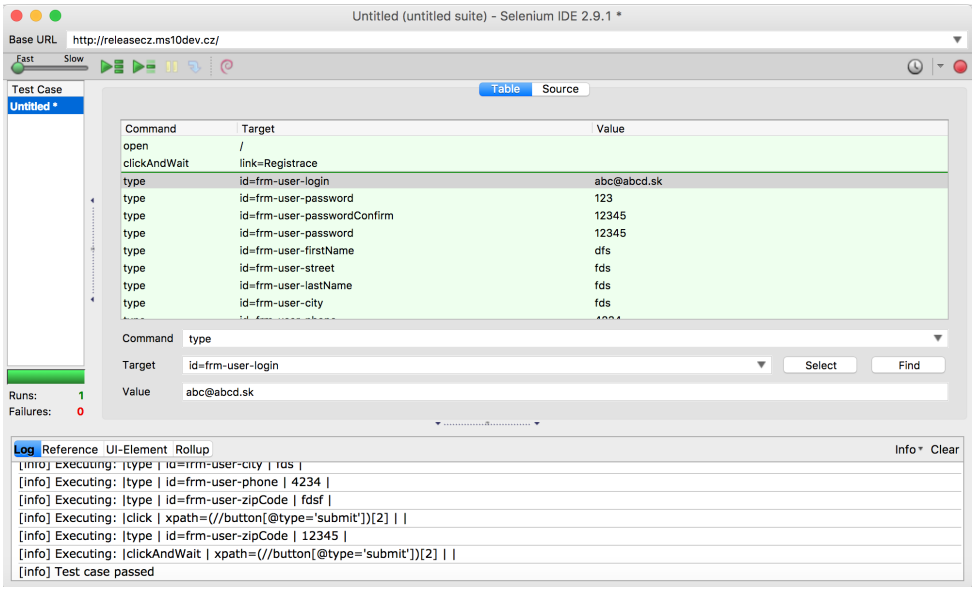
- Síce podporuje automatizované testovanie pre desktopové aplikácie ale len pre WPF aplikácie.

4.2 Selenium

Je to testovací nástroj pre automatizované testovanie webových aplikácií. Je vyvinutý v programovacom jazyku Java a tým pádom máme výhodu, že ho môžeme používať na rôznych platformách. Bol vyvinutý Jasonom Hugginsonom v roku 2004.[8] Selenium sa skladá z viacerých komponentov a každý tento komponent zohráva inú rolu v testovaní. V nasledujúcich podkapitolách si preberieme jednotlivé komponenty.

4.2.1 Selenium IDE

Je to integrované vývojové prostredie, ktoré umožňuje nahrávať, editovať a ladiť automatizované testy. Takto nahrané testy môžeme uložiť vo formáte HTML, ruby skriptu alebo v inom formáte. Tento komponent umožňuje pracovať len s prehliadačom Mozzila Firefox. Na obr. 3 môžeme vidieť ukážku úspešne prebehnutého testu v prostredí Selenium IDE.



Obrázok 3: Selenium IDE[6]

4.2.2 Selenium Remote Control

Je to komponent, ktorý nám ponúka rozsiahlejšie možnosti na vytváranie testov, pretože ich môžeme priamo programovať v nejakom programovacom jazyku. Tieto testy sa následne spúšťajú pomocou Selenium servera, ktorý beží v operačnom systéme. Následne ich spúšťa v nejakom vybranom webovom prehliadači pomocou Selenium Core. Tu máme výhodu v tom,

že nie sme obmedzení len na jeden prehliadač, ako to bolo pri Selenium IDE, ale môžeme použiť rôzne iné prehliadače na spustenie.

4.2.3 Selenium WebDriver

Je to komponent, ktorý je nástupcom komponenta Selenium Remote Control. Na rozdiel od Selenium Remote Control nie je nutné používať na spustenie testov Selenium Server. Umožňuje nám spúšťať testy na viacerých prehliadačoch. Jeho architektúra je jednoduchšia než u jeho predchodcu Selenium Remote Control.

4.2.4 Selenium Grid

Umožňuje nám spúšťať testy na rôznych strojoch prostredníctvom rôznych webových prehliadačov a to paralelne. Podporuje distribuované spustenie testov. Selenium Grid je nástroj, ktorý výrazne urýchľuje funkčné testovanie webových aplikácií.[2]

4.2.5 Výhody

- Hlavnou výhodou tohto nástroja je, že je ponúkaný zadarmo.
- Selenium podporuje napr. na rozdiel od Telerik Test Studia množstvo programovacích jazykov ako Java, Perl, Python, C#, Ruby, Groovy, Java Script a VB Script atď.
- Selenium je multiplatformový, takže ho môžeme spustiť na rôznych operačných systémoch.
- Zaberá len mále množstvo procesora a pamäti pri spustení testov.
- Podporuje množstvo webových prehliadačov ako Internet explorer, Chrome, Firefox, Opera, Safari atď.

4.2.6 Nevýhody

- Keďže je Selenium open-source nástroj, tak neumožňuje žiadnu technickú podporu.
- Selenium podporuje len webovo založené aplikácie.
- Nepodporuje zdieľanie testov a výsledkov.
- Komponent Selenium IDE podporuje len prehliadač Mozzila Firefox.

4.3 TestComplete

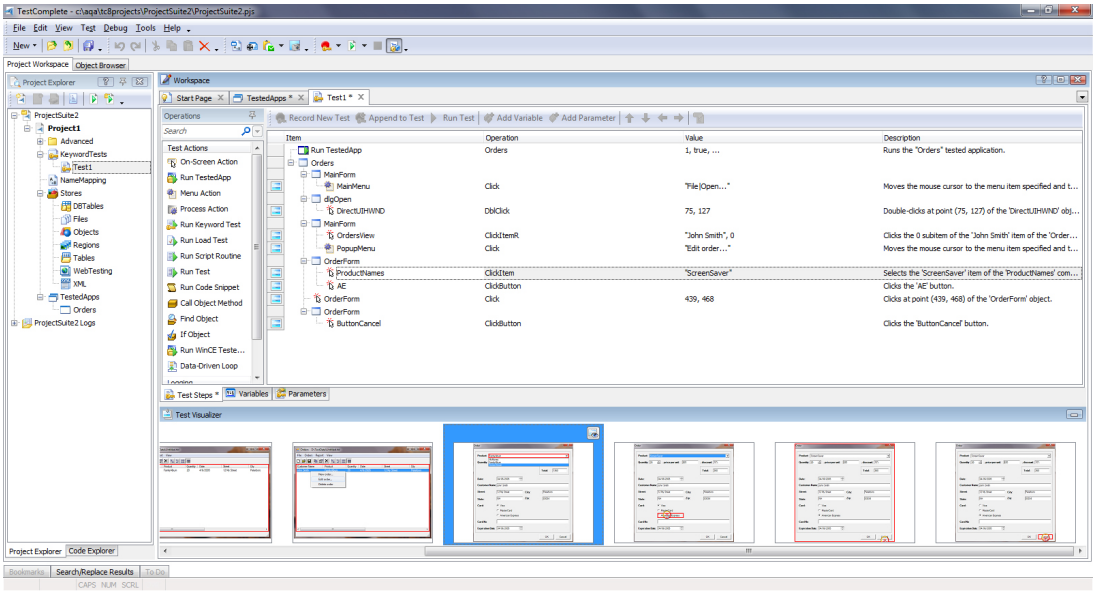
Je to automatizovaný testovací nástroj vyvíjaný spoločnosťou Smart Bear Software pre testovanie webových, desktopových a mobilných aplikácií. TestComplete sa skladá z TestComplete

platformy, ktorá poskytuje testovacie IDE so základnými komponentmi pre automatizované testovanie, nahrávanie, integráciu nástrojov tretích strán a technologické moduly, ktoré rozširujú platformu umožňujúcu testovanie špecifických aplikačných technológií.

Licencia TestComplete je vydávaná v dvoch variantoch[9]:

- Node locked license - v tejto licencii je TestComplete podporovaný len na jednom fyzickom počítači v danej sieti. Cena tejto licencie sa pohybuje od 3 730 €.
- Floating user license - TestComplete je podporovaný na viacerých fyzických alebo virtuálnych počítačoch v danej sieti. Počet súbežných inštancií TestComplete v jednom okamihu nesmie prekročiť počet licencií typu Floating user license. Cena tejto licencie začína od 7 463 €.

Je to testovacie prostredie pre širokú škálu typov aplikácií a technológií, ktoré zahrňujú Windows, .NET, WPF, Visual C++, Visual Basic, Delphi, C++Builder, Java a webové aplikácie a služby. Testy môžeme spúšťať priamo v TestComplete alebo môžu byť exportované do externej aplikácie a tam budú spustené. TestComplete rozpoznáva objekty a ovládacie prvky testovaných aplikácií a ponúka špeciálne príkazy pre simuláciu užívateľských akcií. Ponúka tiež špecifické kontrolne body, ktoré umožňujú ľahko overiť stav aplikácie počas toho, ako bude test spustený. Na obr. 4 môžeme vidieť ukážku testovacieho prostredia nástroja TestComplete.



Obrázok 4: TestComplete[6]

4.3.1 Výhody

- Podporuje viaceré skriptovacie jazyky.

- Umožňuje nahrávanie automatizovaných testov bez toho, aby sme mali skúsenosti so skriptovaním.
- Podporuje taktiež data-driven testovanie a umožňuje využiť externé dátové zdroje ako Excel súbory, CSV súbory, Microsoft SQL Server, Oracle, MySQL, Interbase a ďalšie.
- TestComplete sa dá použiť na rôzne typy testovania: funkčné testovanie, regresné testovanie, jednotkové testovanie, webové testovanie, distribuované testovanie a ďalšie.

4.3.2 Nevýhody

- Podporuje iba operačné systémy Microsoft Windows.
- Nástroj je rovnako ako Telerik Test Studio spoplatnený.

4.4 Celkové zhodnotenie nástrojov

Jednotlivé testovacie nástroje popísané v tejto kapitoly majú svoje určité výhody a nevýhody. Celkové zhrnutie a porovnanie základných vecí môžeme znázorniť do tabuľky. Na tab. 1 môžeme vidieť porovnanie niektorých funkcií spomínaných testovacích nástrojov.

Tabuľka 1: Porovnanie niektorých funkcií testovacích nástrojov

Názov nástroja	**Telerik Test Studio**	**Selenium**	**TestComplete**
Cena	2499$	Zadarmo, Open Source	3730€
Podpora OS	Windows	Multiplatformný	Windows
Podpora programovacích jazykov	C# a VB.NET	Dobrá podpora	Dobrá podpora
Znovupoužitie testov	Áno	Nie	Áno
Testovanie aplikácií	Web, Desktop, Mobile	Iba webové aplikácie	Web, Desktop, Mobile
Zdieľanie testov a výsledkov	Áno	Nie	Áno

Z daného porovnania môžeme usúdiť, že komerčné nástroje ponúkajú na rozdiel od bezplatných nástrojov množstvo rozšírených funkcií. Ďalej je treba zvážiť, či budeme schopní využiť tieto rozšírené funkcie alebo by nám stačili funkcie, ktoré ponúka bezplatný nástroj. Spoločnosť NetDirect, s.r.o. si zvolila Telerik Test Studio ako automatizovaný testovací nástroj pre testovanie systému FastCentrik. Dôvodom bolo, že firma už mala licenciu tohto nástroja zakúpenú, avšak doteraz málo využívanú.

5 Automatizované testovanie systému FastCentrik

Spoločnosť NetDirect, s.r.o. doteraz pre svoj systém FastCentrik používala len manuálne testovanie. Bolo veľmi ťažké testovať všetky funkcionality manuálne, pretože pri veľkom množstve testov to môže testera priviesť k chybe. Najväčšia nevýhoda týchto manuálnych testov je čas vykonávaných testov. Tu prichádzajú na rad automatizované testy, ktoré majú problémy týchto manuálnych testov odstrániť. Nedajú sa síce pokryť všetky možné manuálne testy automatizovanými testami. Tiež sa neodporúča každý jeden manuálny test pokrývať automatizovaným, pretože by bolo veľmi ťažké udržiavať tieto testy a taktiež niektoré testy sa dajú vykonať výhradne len manuálne, pretože pri nich je práve dôležitý ľudský faktor.

Pre vykonávanie automatizovaného testovania systému FastCentrik bol použitý testovací nástroj Telerik Test Studio. Vytvorenie automatizovaných testov spočívalo vo vytvorení testovacích scenárov, ktoré sa skladajú z testovacích prípadov. Jednotlivé testovacie prípady obsahujú kroky pre overenie funkčnosti systému. Najskôr je teda nutné mať vytvorené testovacie prípady a až potom môžeme vytvoriť a implementovať samotný test. Pri vytváraní automatizovaných testov sa kládol dôraz na kritické časti systému.

Systém FastCentrik pozostáva z administračnej časti tzv. backend a z užívateľskej časti tzv. frontend. Administračná časť ponúka funkcionality v oblasti spravovaní produktov, spravovaní objednávok, marketingu, aplikácií 3 strán atď. Užívateľská časť naproti tomu ponúka funkcionality, ktoré prevažne využíva zákazník, jedná sa o objednávanie produktov, filtrovanie produktov, spravovanie užívateľského účtu atď. Jedna aj druhá časť je dôležitá pre úplne fungovanie internetového obchodu, preto je dôležité vytvorenie automatizovaných testov na oboch stranách. V prílohe B je uvedený zoznam funkcionalít užívateľskej a administračnej časti systému, pre ktoré boli vytvorené automatizované testy.

V nasledujúcich podkapitolách je popísaná tvorba automatizovaných testov v testovacom nástroji Telerik Test Studio a v prostredí Visual Studio 2015 a taktiež je znázornená ukážka testovacieho scenára pre administračnú časť a pre užívateľskú časť systému FastCentrik.

5.1 Testovaná aplikácia FastCentrik

Aplikácia FastCentrik je webová aplikácia, ktorá sa skladá z užívateľskej a z administračnej časti. Jej serverová časť je naprogramovaná vo frameworku ASP.NET MVC a klientska časť je naprogramovaná vo frameworku AngularJS. Tento systém je založený na trojvrstvovej architektúre.

Aplikácia FastCentrik je neustále vyvíjaná priebežne. Vývoj prebieha najskôr na vetve Trunk, ak sa otestuje táto vetva a uzná sa, že je bez chyby, tak prejde do vetvy Release. Po otestovaní vetvy Release prejde samotný aktualizovaný produkt do testovania v reálnom prostredí.

Vidíme, že aplikácia FastCentrik prejde určitým testovaním a až potom môže prísť k zákazníkovi. V nástroji Telerik Test Studio si môžeme jednoduchým spôsobom nastaviť definovanú URL

adresu, či už pre Trunk alebo pre Release vetvu. Na obr. 5 môžeme vidieť nastavenie hlavnej URL adresy pre celý Telerik Test Studio projekt.

Obrázok 5: Nastavenie bázovej URL adresy

5.2 Tvorba testov v nástroji Telerik Test Studio

Pri tvorbe automatizovaných testov je možné použiť samostatné rozhranie nástroja Telerik Test Studio alebo integráciu s nástrojom Visual Studio. Pri vykonávaní tejto diplomovej práce sme využili, ako samotné rozhranie Telerik Test Studio, tak aj integráciu s Visual Studiom. Dôvodom je, že Visual Studio poskytuje lepšie nástroje pre implementáciu kódu a samotné rozhranie Telerik Test Studio zase poskytuje lepšie štatistiky vyhodnotenia testov.

Pre vytvorenie automatizovaného testu máme možnosť vytvoriť test cez nahrávanie konkrétne vykonaných krokov v prehliadači, alebo je možné rovno začať s implementáciou kódu. Máme možnosť niektorý nahraný krok editovať, čo môže byť veľmi výhodné, pretože, ak máme nejaký krok, ktorý kontroluje text, tak pri kliknutí na editáciu kroku nás Telerik Test Studio rovno presunie do vývojového prostredia a ukáže nám kód, ktorý vykonáva túto kontrolu textu.

5.2.1 Práca s dátovým zdrojom

Veľmi dôležitou vlastnosťou pre automatizované testovanie sú vstupné dáta. Tieto dáta sú potrebné pre zadávanie vstupov pri konkrétnych krokoch v teste. Môžeme tieto dáta zadávať ručne do nástroja, alebo môžeme k tomu využiť nejaký dátový zdroj. Telerik Test Studio ponúka viaceré dátové zdroje pre uchovanie potrebných dát. V tejto práci sme použili ako zdroj dát XML.

XML zdroj má tú výhodu, že je ľahko čitateľný ako človekom, tak aj počítačom. V nasledujúcom výpise môžeme vidieť ukážku vstupných dát pre test zapamätania si objednávky.

```
<rememberingOrderBaskets>
   <rememberingOrderBasket>
      <products>
         <product>
            <productUrl>/damska-mikina-no-waves-azure-2/d-22</productUrl>
            <numberOfPieces>3</numberOfPieces>
         </product>
         <product>
            <productUrl>/revolution-3</productUrl>
            <numberOfPieces>5</numberOfPieces>
```

```
        </product>
      </products>
   </rememberingOrderBasket>
   <rememberingOrderBasket>
      <products>
         <product>
            <productUrl>/revolution-3</productUrl>
            <numberOfPieces>3</numberOfPieces>
         </product>
         <product>
            <productUrl>/kolecka-gepard/d-44</productUrl>
            <numberOfPieces>5</numberOfPieces>
         </product>
      </products>
   </rememberingOrderBasket>
</rememberingOrderBaskets>
```

Výpis 1: Ukážka vstupných dát pre objednávku

Element **rememberingOrderBasket** vyjadruje iteráciu daného testu. To znamená, že môžeme vytvoriť takýchto elementov toľko, koľko chceme, aby test vykonal počet svojich iterácií. Každý element môže mať iné dáta, čo nám zaručí otestovať funkcionalitu pre rôzne vstupy.

Nevýhodou nástroja Telerik Test Studio je, že nedokáže načítať vnútorné elementy, ktoré tam máme viackrát definované. Príkladom je element **products**, ktorý môže obsahovať viaceré elementy **product**. Elementy **product** vyjadrujú dáta pre konkrétny produkt. Riešením tohto problému je vlastná implementácia načítania týchto dodatočných elementov pri každej iterácií testu.

V nasledujúcom kóde môžeme vidieť inicializáciu konštruktora, ktorý načíta konkrétny xml súbor, ktorý sme pridali v rozhraní Telerik Test Studio. **ExecutionContext** je kontext aktuálne spusteného testu a umožňuje nám zistiť, ktorý dátový súbor je aktuálne pripojený k testu.

```
public string XmlFilePath { get; set; }
public XML(ExecutionContext context)
{
  XmlFilePath = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)
      .Replace("bin", "Data\\" + context.Test.Source + "");
}
```

Výpis 2: Konštruktor pre načítanie XML dát

V našich jednotlivých testoch, kde potrebujeme pracovať s takýmito typmi elementov, tak je potrebné zavolať metódu, ktorá získa tieto potrebné hodnoty z elementov. V nasledujúcej

metóde je ukázané získanie hodnôt z elementov. Je potrebne definovať parameter **index**, ktorému predáme aktuálny index iterácie. Následne je potrebné definovať parameter **rootElement** a **targetElement**, čo je názov koreňového elementu a cieľového elementu. V našom xml súbore ide o elementy products a product.

```
public IEnumerable<XElement> GetItemsFromXmlElement(int index, string
    rootElement, string targetElement)
{
   XElement e = XElement.Load(XmlFilePath);
   IEnumerable<XElement> data = e.Descendants(rootElement).ElementAt(index).
      Element(targetElement).Elements();
   return data;
}
```

Výpis 3: Metóda načítania hodnôt xml elementu

5.2.2 Pomocné testovacie triedy

Telerik Test Studio nám umožňuje vytvárať okrem samotných testovacích tried aj normálne C# triedy, ktoré sú ideálne pre doplnkovú funkcionalitu pre naše automatizované testy. V tejto práci sme vytvorili viacero pomocných tried, ktoré nám uľahčili tvorbu testov. Je to napr. pomocná statická trieda **HelperFunctions**, ktorá obsahuje pomocné funkcie pre parsovanie, validovanie e-mailovej adresy, kontrolu existencie elementu.

Máme, napr. tiež statickú triedu **Config**, ktorá sa stará o nastavenie konfigurácie určitých vecí pre test. Napr. pri získaní URL adresy umožňuje Telerik Test Studio API získať len bázovú URL adresu aplikácie, avšak my potrebujeme adresu pre administračnú časť a užívateľskú časť. Pomocná môže byť teda metóda, ktorá nám nastaví tieto veci. V nasledujúcej ukážke kódu vidíme funkciu, ktorá nastaví bázovú URL adresu na administračnú adresu:

```
public static string GetBackendUrl(string baseUrl)
        {
            string resultBaseUrl = string.Empty;
            if (!baseUrl.Contains("admin"))
            {
                string [] split = baseUrl.Split(new string[]{"//"},
                    StringSplitOptions.None);
                string prefix = "admin.";
                resultBaseUrl = split[0]+"//"+prefix+split[1];
            }
            else
            {
```

```
            resultBaseUrl = baseUrl;
        }
        return resultBaseUrl;
    }
```

Výpis 4: Metóda nastavenia administračnej URL adresy

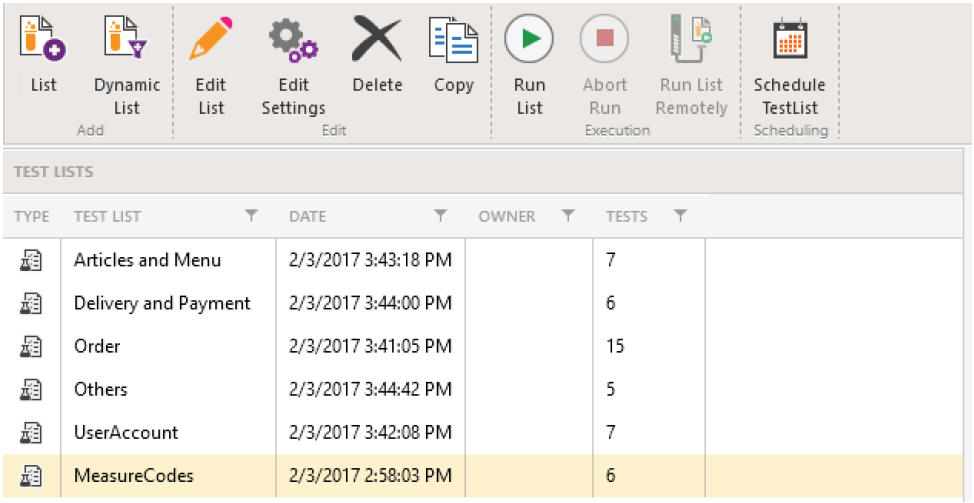
5.2.3 Pomocné testy

Pri vytváraní ďalších automatizovaných testov sa môžeme stretnúť so situáciou, kedy nejakú časť, ktorú sme vykonali v minulom teste, potrebujeme vykonať i v tom aktuálnom. Telerik Test Studio nám umožňuje spájať vytvorené testy do iných testov. Týmto sa stáva samotné testovanie efektívnejšie a znovupoužiteľnejšie.

Príkladom môže byť test na prihlásenie do administrácie. Každú funkcionalitu, ktorú chceme v administrácii otestovať, vyžaduje prihlásenie. Preto môžeme vytvoriť jeden test na prihlásenie a pripájať ho do ostatných testov, ktoré ho vyžadujú. Týmto zamedzíme duplicitnému vytvoreniu a tiež sú naše testy efektívnejšie a znovupoužiteľnejšie.

5.2.4 Testovacie zoznamy

Veľkou výhodou nástroja Telerik Test Studio je, že ponúka spustenie automatizovaných testov prostredníctvom testovacích zoznamov tzv. Test lists. Tieto testovacie zoznamy nám veľmi zjednodušia prácu vtedy, ak máme veľa testov a nechceme ich sekvenčne spúšťať jednotlivo. Umožňuje nám ich vykonať všetky jedným spustením. Tiež nám Telerik Test Studio umožňuje tieto testovacie zoznamy spúšťať prostredníctvom vzdialeného stroja. Taktiež ponúka Telerik Test Studio spustiť testovacie zoznamy v definovaných časových intervaloch. Takýto postup spustenia testovacích zoznamov je výhodný, pretože si môžeme napr. naplánovať spustenie testovacích zoznamov cez noc a nebyť počas dňa nimi obmedzení. Na obr. 6 môžeme vidieť konkrétnu ukážku testovacích zoznamov pre systém FastCentrik.



Obrázok 6: Testovacie zoznamy pre systém FastCentrik

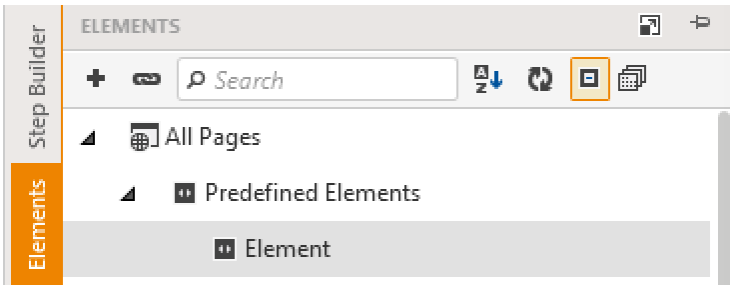
Jednotlivé tieto testovacie zoznamy je možné spúšťať súhrnne naraz a tak isto aj po jednom. Výhodou je, že ak chceme otestovať len konkrétnu kategóriu funkcionalít, tak nemusíme zbytočne spúšťať všetky testovacie zoznamy. Tak isto je možnosť kedykoľvek si vytvoriť nový testovací zoznam a pridať do neho testy podľa potreby.

5.3 Prieskumník elementov

Telerik Test Studio nám ponúka funkciu na preskúmanie HTML elementov. Je to podobné ako DOM prieskumník, avšak s tým rozdielom, že obsahuje len tie elementy, ktoré používame v teste. Viaceré elementy môžu byť použité vo viacerých testoch, avšak iba raz je zobrazený v prieskumníku.

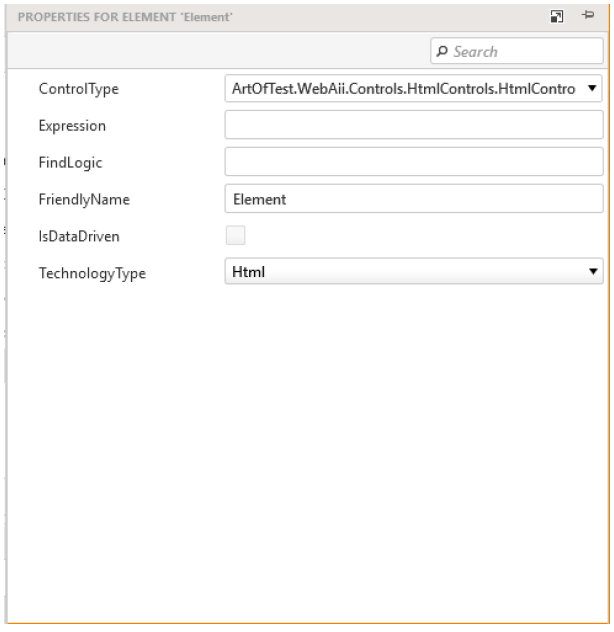
Máme možnosť si vytvoriť aj svoje vlastné preddefinované elementy, ktoré sú výhodné vtedy, ak by sme museli čakať na dokončenie funkcionality, aby sme ju mohli otestovať. Tieto preddefinované elementy si môžeme neskôr, ak už bude funkcionalita dokončená, napojiť na aktuálne elementy na danej stránke.

Na obr. 7 môžeme vidieť ukážku pridania preddefinovaného elementu. Môžeme si vytvoriť aj viac elementov naraz a potom ich jednotlivo priradzovať aktuálnym elementom.



Obrázok 7: Pridanie preddefinovaného elementu

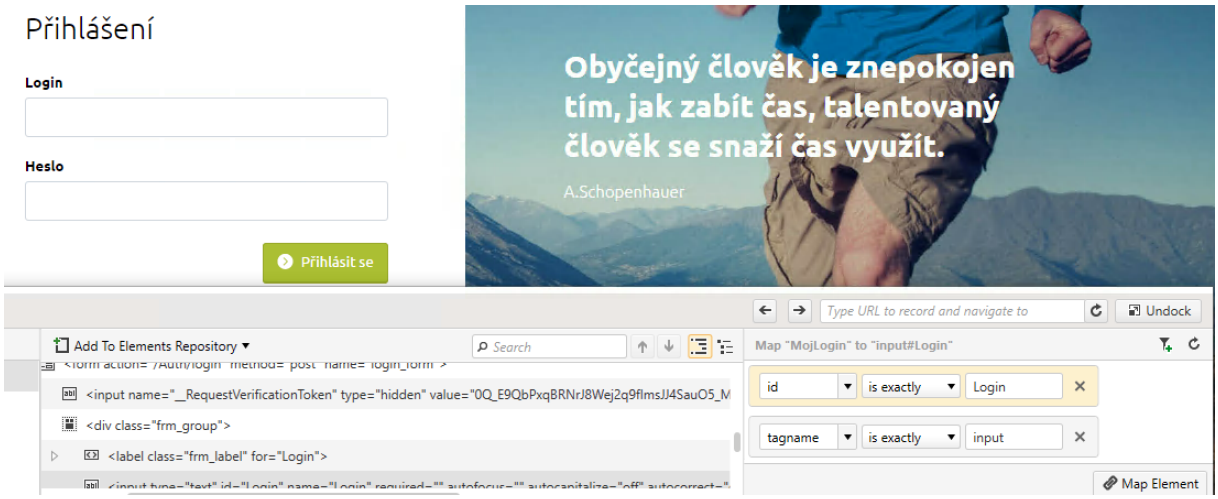
V rámci vytvorenia preddefinovaného elementu mu môžeme popridávať rôzne vlastnosti. Môžeme stanoviť, že ide o element, ktorý bude typu **HtmlControl**, čo je bázový typ pre všetky typy elementov. Môžeme stanoviť, či je daný element napojený na dátový zdroj a tiež je dôležité stanoviť názov elementu, ktorý bude zobrazený v strome elementov v Telerik Test Studiu. S týmto názvom budeme pracovať, ak si necháme daný element vygenerovať ako kód. Odporúča sa používať názvy bez diakritiky kvôli lepšej prehľadnosti a použiteľnosti. Na obr. 8 vidíme nastavenie vlastností preddefinovaného elementu.



Obrázok 8: Nastavenie vlastností preddefinovaného elementu

Ak si vytvoríme a prichystáme tieto elementy pred tým, než bude funkcionalita od vývojárov dokončená, tak potom môžeme tieto elementy napojiť na aktuálne elementy, ktoré sa nachádzajú na daných stránkach. Najskôr spustíme prehliadač a potom môžeme vybrať element, ktorý chceme mapovať na vytvorený preddefinovaný element. Na obr. 9 môžeme vidieť mapova-

nie elementu Login na náš vytvorený preddefinovaný element. Pri mapovaní sa nám zobrazí aj vyhľadávacia logika pre tento element. Môžeme si nastaviť aj svoju vlastnú vyhľadávaciu logiku pre tento element.



Obrázok 9: Mapovanie preddefinovaného elementu na aktuálny element

Výhodou nástroja Telerik Test Studio je, že ak testujeme prostredníctvom nahrávania obrazovky a chceme si nejaký element označiť a priradiť mu nejakú udalosť, tak automaticky sa nám zobrazí v strome prieskumníka elementov. Telerik Test Studio mu automaticky vygeneruje vyhľadávaciu logiku a môžeme si túto logiku aj neskôr zmeniť. Ak by sme chceli príslušnú vyhľadávaciu logiku zmeniť, tak máme možnosť po nastavení vyhľadávacieho filtra prostredníctvom jediného tlačidla otestovať, či bude daný element nájdený alebo nie. Týmto spôsobom budeme mať plnú kontrolu nad elementmi.

5.4 Testovacie techniky

Pri každom testovaní nejakej funkcionality chceme dosiahnuť, aby naše testovanie odhalilo akúkoľvek chybu. Ak by sme vytvárali naše testy dosadením ľubovoľných dát, tak nemali by sme vždy istotu, že sme na niečo nezabudli alebo, že sme zbytočne dosadili rovnaké dáta ako v predchádzajúcom teste.

Na to, aby sme tieto problémy vstupných dát obišli, tak existujú testovacie techniky. Existuje veľa rôznych testovacích technik, niektoré sú špecifické pre testovanie čiernej skrinky a iné pre testovanie bielej skrinky. V tejto práci vykonávame automatizované testovanie, tak sme zameraní na testovanie čiernej skrinky, pretože nemáme k dispozícii zdrojový kód, ktorý by nám viac o danej aplikácii povedal.

Čo sa týka testovania čiernej skrinky, tak poznáme, napr. tieto testovacie techniky:

- ekvivalentné rozdelenie,
- analýza hraničných hodnôt,

- testovanie prechodov medzi stavmi,
- graf príčin a následkov.

Každá z týchto technik je niečím špecifická a plní svoj účel. Pri tejto práci budeme využívať techniky ekvivalentné rozdelenie a analýzu hraničných hodnôt.

5.4.1 Ekvivalentné rozdelenie

Ekvivalentné rozdelenie je technika testovania čiernej skrinky, ktorá rozdeľuje vstup aplikácie do tried. Každá trieda nám dáva rovnaký výsledok. To znamená, že táto technika nám redukuje počet testovacích prípadov. Bez tejto techniky by sme vytvárali veľké množstvo testovacích prípadov, ale s použitím techniky nám stačí vytvoriť jeden testovací prípad z každej ekvivalentnej triedy.

V tejto práci sme využili túto techniku na formulár registrácie. Ukážeme si jeden príklad aplikovania tejto techniky pre vstup zadávania **mesta**. Pri skúmaní tohto vstupného políčka sme zistili, že akceptuje zadanie mesta s aspoň 1 znakom a maximálne so 100 znakmi. Tento vstupný parameter si môžeme rozdeliť na 3 triedy, ktoré vidíme znázornené na tab. 2. Na tomto obrázku vidíme, že nám stačí vytvoriť 3 testovacie prípady.

Prvý testovací prípad zahrňuje vstupy, ktoré sú menšie ako 1 znak, ide o vstup, ktorý neobsahuje žiadne znaky a je teda neplatný. Druhý testovací prípad zahrňuje vstupy s počtom znakov od 1 do 100 znakov. V tomto testovacom prípade si môžeme vybrať ľubovoľný počet znakov v tomto intervale a budeme predpokladať, že aj ostatné hodnoty znakov v tomto intervale nám dajú rovnaký výsledok, čiže v tomto prípade ide o platný vstup. Tretí testovací prípad zahrňuje znaky, ktoré sú väčšie ako 100 znakov. Ide opäť o neplatný vstup a môžeme si vybrať ktorúkoľvek hodnotu znakov v tomto intervale.

Tabuľka 2: Ekvivalentné rozdelenie vstupu mesta do tried

Triedy ekvivalencie	**< 1**	**1 - 100**	**> 100**
Platný/Neplatný vstup	Neplatný	Platný	Neplatný
Vybraná hodnota	0	5	120

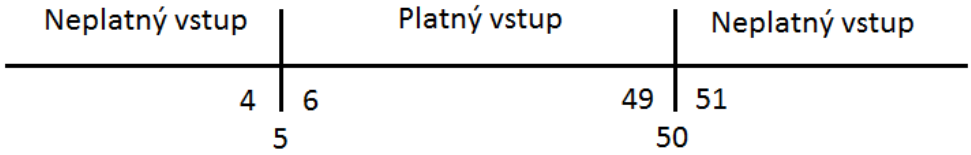
5.4.2 Analýza hraničných hodnôt

Analýza hraničných hodnôt je testovacia technika, ktorá je rovnako ako technika ekvivalentného rozdelenia zameraná na testovanie čiernej skrinky. Táto technika umožňuje testovať hraničné hodnoty medzi ekvivalentnými časťami. Je rozšírením techniky ekvivalentného rozdelenia. Umožňuje nám nájsť viac typov chýb než technika ekvivalentného rozdelenia, avšak musíme vytvoriť viac testovacích prípadov, než u ekvivalentného rozdelenia.

Táto technika spočíva v tom, že, ak sme si vstup aplikácie rozdelili do ekvivalentných tried, tak pre každú hranicu vytvoríme dva testovacie prípady. Jeden aj druhý prípad budú na každej

strane hranice čo najbližšie. V tejto práci sme túto techniku použili rovnako ako u ekvivalentného rozdelenia na registračný formulár. Ukážeme si príklad na vstupný parameter zadávania **hesla**. Pri tomto hesle neboli pri vstupe žiadne veľké obmedzenia. Jediným obmedzením bola stanovená dĺžka hesla. Vstup musí akceptovať minimálne 5 znakov a maximálne 50 znakov.

Na obr. 10 vidíme aplikovanú techniku analýzy hraničných hodnôt pre vstup zadávania hesla, kde máme identifikované hranice vstupov. Najskôr sme si definovali tri ekvivalentné triedy a potom sme na hraniciach s týmito triedami určili hraničné hodnoty. Určili sme, že výsledných bude šesť testovacích prípadov.



Obrázok 10: Analýza hraničných hodnôt pre vstup hesla

Dva testovacie prípady definujú minimálne a maximálne platné hraničné hodnoty (5;50). Ďalšie dva testovacie prípady definujú hodnotu tesne pod/nad hraničnou hodnotou pre neplatný vstup (4;51) a posledné dva testovacie prípady definujú hodnotu tesne pod/nad hraničnou hodnotou pre platný vstup (6;49). Týmto spôsobom máme síce viac testovacích prípadov, ako u ekvivalentného rozdelenia, avšak zabezpečuje nám detailnejšie odhalenie chýb.

5.5 Testovacie scenáre systému FastCentrik

Jednotlivé testovacie scenáre, ktoré boli vytvorené pre systém FastCentrik nám definujú funkcionality, ktoré sa majú otestovať. Tieto scenáre nám nehovoria, ako sa tieto funkcionality budú testovať. O tom, ako sa budú tieto funkcionality testovať, nám hovoria testovacie prípady.

Každý testovací scenár môže mať jeden a viac testovacích prípadov. V nasledujúcich podkapitolách si ukážeme ukážku testovacích scenárov pre administračnú a užívateľskú časť systému. Predtým, než si tieto príklady ukážeme, si popíšeme, z ktorých častí pozostáva testovací scenár a testovací prípad.

Testovací scenár je v podstate jednoriadková informácia o tom, čo sa bude testovať. Testovací prípad už hovorí o tom, ako sa to bude testovať. Testovací prípad pozostáva z nasledujúcich častí:

- **ID** - určuje nám jedinečný identifikátor testovacieho prípadu.
- **Popis** - v krátkosti nám popisuje, ktorú funkcionalitu bude testovaci prípad pokrývať.

- **Predpoklady** - sú to podmienky, ktoré sú potrebné na to, aby mohol test prebehnúť v poriadku.
- **Kroky** - očíslované kroky, ktoré nám hovoria, čo presne musí tester vykonať, napr. zadaj hodnotu "text"do políčka.
- **Očakávané výsledky** - sú to výsledky, ktoré očakávame, že nastanú po vykonaní krokov.

Všetky tieto časti sú základnými časťami testovacieho prípadu. Hoci obsahuje testovací prípad i množstvo ďalších časti, my si ešte uvedieme ďalšie dve špecifické časti pre naše testovacie prípady:

- **Dáta** - určujú cestu k presným dátam pre konkrétny testovací prípad. V tejto práci sme použili pre XML ako dátový zdroj pre uloženie našich dát. Preto cesta k dátam v tomto prípade je cesta ku konkrétnemu elementu z koreňového elementu.
- **Testovací skript** - určuje presnú cestu súboru testovacieho skriptu v testovacom nástroji. Táto vlastnosť je vhodná vtedy, ak už ide o väčšie množstvo automatizovaných testov, a aby sme vedeli, ktorý testovací skript patrí, ku ktorému testovaciemu scenáru a prípadu.

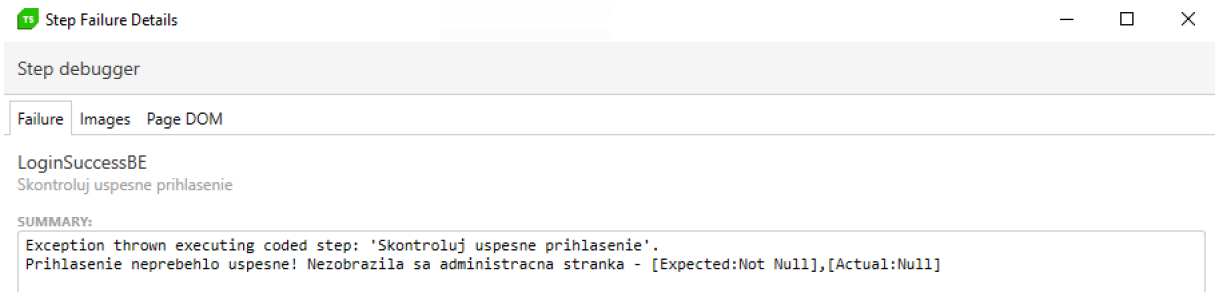
5.5.1 Ukážka testovacieho scenára

Ako ukážku testovacieho scenára si zoberieme funkcionalitu pre užívateľskú časť systému. Ide o funkčnosť objednávania produktov registrovaným zákazníkom v jednokrokovej objednávke. Tento scenár obsahuje jeden testovací prípad a to úspešné objednávanie produktov. Samotnú ukážku testovacieho prípadu môžeme vidieť v prílohe F na konci textu.

Pri tomto teste môžeme vidieť, že niektorý krok sa odkazuje na ID iného testovacieho prípadu, aby sa zbytočné nemuseli opakovať rovnaké kroky. Vidíme tiež, že test obsahuje aj spomínané pridané časti, ktoré sú špecifické pre naše testovacie prípady a umožňujú samotnému testerovi jednoducho sa zorientovať. Tester po prečítaní má možnosť presne zistiť, kde presne sa na dáta odkazovať a ktorý testovací skript vykonáva tento testovací prípad. Všetky ostatné testovacie prípady pre ďalšie funkcionality sú obsiahnuté v priloženom CD.

5.6 Vyhodnotenie výsledkov testu

Výhodou automatizovaných testov je, že sa vykonávajú v testovacom nástroji, ktoré nám ponúka výborne detailnú štatistiku o výsledku testu. Záleží od samotného nástroja, čo všetko nám dokáže zobraziť, ale prostredníctvom nástroja Telerik Test Studio máme bohatú štatistiku. Pri každom spustení testu dostaneme po jej dokončení správu, v ktorej môžeme nájsť, napr. príčinu chyby, kvôli ktorej test zlyhal. Na obr. 11 môžeme vidieť ukážku správy, ktorá sa nám zobrazila po dokončení testu na prihlásenie užívateľa do administrácie. V tomto teste sme úmyselne zadali nesprávne heslo, aby sme mohli ukázať, ako sa nám zobrazí chyba.



Obrázok 11: Správa o dokončení testu na prihlásenie do administrácie

Je dobre používať v testoch verifikácie, aby sme vedeli po dokončení testu, prečo test skončil danou chybou. Na tomto teste používame na verifikovanie úspešného prihlásenia triedu **Assert** a jej metódu **isNotNull**, ktorá nám zaručí, že element, ktorý v sebe zahŕňa administračnú stránku existuje.

```
Assert.IsNotNull(adminPage, "Prihlasenie neprebehlo uspesne! Nezobrazila sa
    administracna stranka");
```

Výpis 5: Verifikovanie úspešného prihlásenia do administrácie

V niektorých prípadoch si nevystačíme so samotnou triedou **Assert**, ale bude nutné ešte dodatočným spôsobom oznámiť výskyt chyby. Tieto prípady nastávajú vtedy, ak nejaký element potrebuje čakať na zobrazenie. Problémom je, že, ak tomuto elementu skončí doba na čakanie, tak vyhodí výnimku. Tým, že vyhodí výnimku a mali by sme ho v triede **Assert**, tak sa nám aj tak nespustí verifikácia. Riešením je v takomto prípade vyhodiť vlastnú výnimku, ak ide o čakanie elementov.

6 Rozdelenie aplikácie podľa kritickosti

Systém FastCentrik pozostáva z rozličných funkcií, niektoré sú dostupné pre administrátora systému a iné pre zákazníka, ktorý objednáva produkty. Tieto funkcie boli zaradené do priorít podľa viacerých faktorov. Priorita je uprednostňovanie funkcií podľa dôležitostí. Táto dôležitosť môže byť stanovená podľa toho, ako veľmi je daná funkcia dôležitá pre firmu, ako veľmi by chyba v tejto funkcii mala dopad na ostatné funkcie a ako často sú tieto funkcie využívané zákazníkmi.

Jednotlivé funkčnosti systému FastCentrik sú kategorizované do štyroch priorít a sú rozdelené ako pre užívateľskú časť, tak aj pre administračnú časť systému. V nasledujúcom zozname máme stručný popis všetkých štyroch priorít:

- Priorita 1 - patria tu najkritickejšie časti systému, bez ktorých systém nedokáže reálne fungovať a chyba v tejto časti má fatálne následky pre celú firmu. Systém nemôže byť použitý, dokiaľ nebudú chyby v týchto častiach opravené. Tieto chyby by mali byť odstránené tak rýchlo, ako je to len možné. Všetko ostatné by malo ísť bokom a mali by sa tieto chyby len riešiť.

- Priorita 2 - patria tu funkčnosti, ktoré sú o niečo menej kritickejšie než v prvej priorite. Avšak, ak nastane v nich chyba, tak musí byť vyriešená okamžitá, pretože táto chyba ovplyvňuje systém do značnej miery a príslušné moduly nemožno použiť, dokiaľ nie sú opravené.

- Priorita 3 - patria tú funkčnosti, pri ktorých, ak nastane chyba, tak ju stačí riešiť až v nasledujúcom zostavení aplikácie.

- Priorita 4 - do tejto poslednej priority patria funkčnosti, ktoré zákazníci až tak často nepoužívajú a v prípade chyby ju stačí riešiť, až pokým máme vyriešené ostatné časti vo vyšších prioritách. Pre tieto funkčnosti nie je nutné vytvárať automatizované testy, pretože sa tieto časti málo používajú.

Rozdelenie jednotlivých funkčnosti podľa priorít pre užívateľskú časť môžeme nájsť v prílohe G a pre administračnú časť nájdeme v prílohe H. Tieto rozdelenia budú v ďalšej kapitole znázornené do mapy pokrytia, ktorá prináša detailnejší pohľad a poskytuje lepšiu orientáciu v pokrytí testov.

6.1 Odporúčania pre zlepšenie prioritizácie testovacích funkčností

Okrem samotného posúdenia, ktoré časti systému sú prioritnejšie, a ktoré menej prioritnejšie, poznáme aj rôzne iné metódy, na základe ktorých by sme mohli dané funkčnosti rozdeliť. Jedna z týchto metód sa nazýva **Multidimenzionálna analýza rizík**.

Táto metóda je nástrojom, ktorý môžeme aplikovať na mnoho zložitých rozhodnutí. Je to teda najviac využiteľné pri riešení problémov, ktoré sú charakterizované ako voľby medzi alter-

natívami. Pomáha nám zamerať sa na to, čo je logické, konzistentné a ľahko použiteľné. Táto metóda je užitočná pre[11]:

- delenie rozhodnutí na menšie celky, lepšie porozumenie jednotlivých častí,
- analyzovanie jednotlivých častí,
- integrovanie častí pre vytvorenie zmysluplného diela.

Existujú rôzne metódy multidimenzionálnej analýzy rizík. V nasledujúcom zozname uvedieme krátky výber daných metód:

- analytický hierarchický proces,
- analytický sieťový proces,
- analýza dátových obalov,
- hodnotová analýza.

V rámci týchto metód by sme mohli pre našu prioritizáciu použiť, napr. metódu analytického hierarchického procesu. Táto metóda je štruktúrovanou technikou, ktorá si dokáže poradiť s komplexnými rozhodnutiami a môže nám pomôcť nastaviť správne priority a vybrať tú najlepšiu voľbu. Metóda bola vyvinutá Thomasom L. Saatyom v roku 1970. Algoritmus tejto metódy je štruktúrovaný do dvoch krokov:

- určenie relatívnych váh rozhodujúcich kritérií,
- určenie relatívnych priorít alternatív.

7 Mapa pokrytia aplikácie

Celkové znázornenie vytvorených automatizovaných testov rozdelených podľa kritickosti je výhodné znázorniť do mapy, ktorá môže slúžiť ako podklad pre firmu, aby vedela, ktoré testy sú pokryté, a ktoré ešte nie. Tiež môže byť táto mapa výhodná pre ostatných testerov, ktorí budú môcť jednoducho sa zorientovať v rámci všetkých otestovaných funkčností. V tejto diplomovej práci som vytvoril 21 automatizovaných testov pre administračnú časť a 23 automatizovaných testov pre užívateľskú časť. Zvyšok funkčností, ktoré vidíme na znázornených mapách, urobil druhý kolega zo spoločnosti NetDirect, s.r.o.

V nasledujúcom zozname uvedieme pokrytie v percentách v rámci jednotlivých priorít:

- Priorita 1 - zo všetkých 32 najkritickejších častí bolo pokrytých 100 %. Zabezpečili sme teda, že pre každú najviac kritickú časť existuje automatizovaný test a dokážeme tak našimi testami zabrániť fatálnym následkom, ktoré by mohli vzniknúť.
- Priorita 2 - zo všetkých 29 kritických častí s prioritou 2 sme pokryli 25 funkčností, to znamená, že celkové pokrytie činí 84 %.
- Priorita 3 - zo všetkých 29 menej kritických častí sme pokryli 24 funkčnosti, to znamená, že celkové pokrytie činí 82,7 %.
- Priorita 4 - zo všetkých 7 najmenej kritických častí sme pokryli 3 funkčnosti, to znamená, že celkové pokrytie činí 42,8 %.

Nedá sa pokryť každá možná funkcionalita v aplikácií automatizovaným testom, pretože by to bolo z hľadiska času veľmi náročné a tiež by to bolo veľmi náročné na údržbu. V prílohách C a D na konci textu môžeme vidieť mapy pre administračnú časť a užívateľskú časť. Mapy boli vytvorené v bezplatnom nástroji XMind. Na mapách môžeme vidieť, že obsahujú prepojenie funkcionalít z hľadiska toho, ako sa jednotlivé funkcionality vykonávajú. Na obidvoch mapách môžeme vidieť, že sú niektoré funkcionality, pre ktoré ešte nie sú vytvorené automatizované testy. Je to preto, lebo majú nižšiu prioritu a museli sme sa sústrediť na tie najkritickejšie časti. Legendu pre jednotlivé symboly na mape môžeme nájsť v prílohe E na konci textu.

8 Odporúčania písania programového kódu

Pri vytváraní automatizovaných testov nad systémom FastCentrik sme sa stretli s viacerými problémami, ktoré nám bránili v tom, aby sme mohli vytvoriť automatizovaný test bezproblémovo. Často tieto problémy viedli k tomu, že niekedy sme testy bohužiaľ museli písať zložitejším spôsobom, aby mohli danú funkcionalitu naplno skontrolovať. V nasledujúcej podkapitole budú uvedené všetky problémy a ich riešenia, pri ktorých sme narazili pri systéme FastCentrik.

8.1 Problém pripojenia elementov

Často sa vyskytujúcim problémom pri automatizovanom testovaní bolo, že niektorá stránka alebo formulár nemali žiadne koreňové elementy, ktoré by obsahovali **id** atribút. Týmto spôsobom bolo ťažké pripojiť sa na koreňový element tak, aby sme mohli dosiahnuť väčšiu znovupoužiteľnosť testu. Muselo sa pristúpiť k alternatívnemu riešeniu a to k pripojeniu sa na štruktúru stránky alebo na **class** atribúty.

Do budúcnosti by bolo vhodnejšie, keby každý nejaký rodičovský element obsahoval nejaký id atribút, pretože tým môžu byť testy ľahšie na správu a stanú sa viac znovupoužiteľnými.

8.2 Problém prázdnych znakov

Pri niektorých automatizovaných testoch sa nedá vždy držať len HTML **id** atribútov, preto musíme použiť aj menej používajúce atribúty ako **class** atribúty. Pri niektorých stránkach systému FastCentrik došlo k problému, že tieto **class** atribúty obsahovali prázdne medzery, čo môže skomplikovať tvorbu testu a znovupoužiteľnosť testu. Príklad takéhoto problému môžeme vidieť v nasledujúcej ukážke:

```
<dl class=" withoutVat "><dt>Cena bez DPH</dt><dd>225,00 Kc</dd></dl>
```

Vidíme, že hodnota v atribúte **class** obsahuje na začiatku aj na konci prázdne medzery a tým sa nemôžeme jednoducho pripojiť na tento element prostredníctvom telerik príkazu:

```
ActiveBrowser.Find.ByXpath("//*[@class,'withoutVat']")
```

Riešením takéhoto problému je napojenie sa na iný element alebo v najhoršom prípade môžeme použiť konštrukciu **contains** jazyka XPath, ktorá nehľadá element podľa presne zadanej hodnoty ale podľa toho, či obsahuje nejaké znaky z hodnoty daného elementu.

8.3 Problém hodnôt cien

Pri testovaní objednávacieho procesu je potrebné otestovať, či sú hodnoty cien správne prepočítané a či sa rovnako zobrazujú v ďalších krokoch objednávky. Pri tomto postupe je teda nutné získať hodnoty týchto cien a pracovať s nimi. Avšak pri niektorých cenách bolo ťažké získať hodnotu a pracovať s ňou, pretože bolo nutné reťazec najskôr rozparsovať a následne

z neho získať len číselnú hodnotu. Príklad takéhoto problému môžeme vidieť v nasledujúcom HTML kóde:

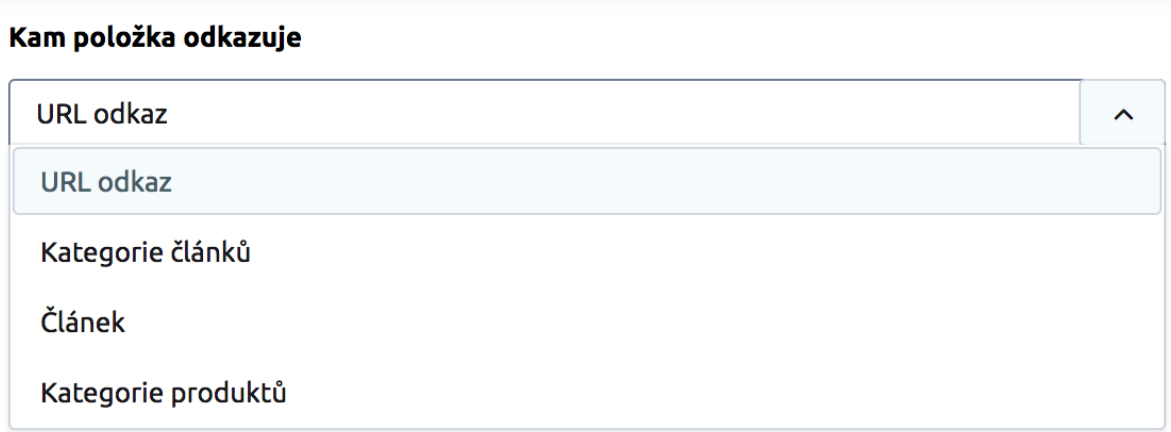
```
<dd>"31,0 Kc"<span class="vatSuffix">s DPH</span></dd>
```

V uvedenom kóde vidíme, že cena je zadaná ako text spolu s menou. Optimálnejším riešením než rozparsovanie a získanie hodnôt by bolo vložiť nejaký pomocný atribút, v ktorom by bola čisto len číselná hodnota, a takto pre človeka, ktorý vytvára test, by bolo jednoduchšie sa na tento atribút odkázať.

8.4 Problém spracovania hodnôt elementu

Pri niektorých testoch bolo potrebné pracovať s hodnotami zo select elementu. Príkladom v systéme FastCentrik môže byť testovanie vytvorenia menu v administračnej časti. Pri tomto vytvorení vyberáme z výberového políčka hodnotu, ktorá špecifikuje, či sa jedná o menu pre URL odkaz, článok, kategóriu článkov alebo kategóriu produktov.

Na obr. 12 môžeme vidieť konkrétnu ukážku zo systému FastCentrik, v ktorej vidíme, že máme na výber tie štyri spomínané hodnoty.



Obrázok 12: Položky pre výber menu

Avšak, pri každej tvorbe testu, musíme dbať na to, aby sme boli odolní voči nejakým zmenám vykonaným v danej aplikácií. Ak sa pozrieme na nasledujúci zdrojový kód tohto elementu, tak môžeme vidieť, že obsahuje len hodnoty týchto položiek, ale nemáme žiadny iný záchytný bod, ktorý by nám umožňoval docieliť znovupoužitie.

```
<div class="selectize-dropdown-content"><div class="option selected" data-
    selectable="" data-value="0">URL odkaz</div><div class="option" data-
    selectable="" data-value="101">Kategorie clanku</div><div class="option"
    data-selectable="" data-value="100">Clanek</div><div class="option active"
    data-selectable="" data-value="202">Kategorie produktu</div></div>
```

Aplikácia FastCentrik je viacjazyková aplikácia, tak to znamená, že, ak by sme sa odkazovali len na textové hodnoty, tak by sme museli náš test opraviť. Tento problém môžeme vyriešiť tak, že sa budeme odkazovať na elementy na základe poradia hodnôt. Avšak ani táto alternatíva nie je 100 %, pretože sa môže ľahko stať, že vývojár zmení poradie položiek. Ideálnym riešením by bolo, ak by sme v každom elemente mali nejaký atribút, ktorý by definoval názov tejto menu položky a tieto atribúty by boli platné aj pri iných jazykoch aplikácie. Týmto spôsobom by sa naše testovanie stalo odolným, prehľadnejším a efektívnejším.

9 Proces zostavenia

Každá softvérová aplikácia je vydávaná vo forme zostavení, ktoré sa anglicky nazývajú **build**. Toto zostavenie zahŕňa v sebe kompilovanie zdrojového kódu, registráciu DLL knižníc, kopírovanie a presúvanie súborov a následne nasadenie systému. Každé zostavenie softvéru je identifikované verziou zostavenia. Spoločnosť NetDirect, s.r.o. v priemere vykonáva každý deň približne 5 zostavení.

Samotné výsledné zostavenie nezaručuje, že softvér funguje bez chyby a správne. Preto existuje priebežná integrácia, ktorá umožňuje tento proces zautomatizovať. Priebežná integrácia je metóda vývoja softvéru, pri ktorej členovia tímu priebežne integrujú svoju prácu, najlepšie sa odporúča aspoň raz denne. V rámci integrácie je preto výhodné pridať automatizované testy, ktoré nám zaručia menšiu chybovosť softvéru a taktiež menší počet vykonávaných zostavení. Pre potrebu zavedenia automatizovaných testov do procesu zostavenia sme použili nástroj TFS od spoločnosti Microsoft. Cieľom v tejto práci bolo zaviesť vytvorené automatizované testy do procesu zostavenia, aby odhalenie chýb bolo presnejšie, a aby prebehlo ešte pred samotným nasadením na server.

9.1 Team Foundation Server

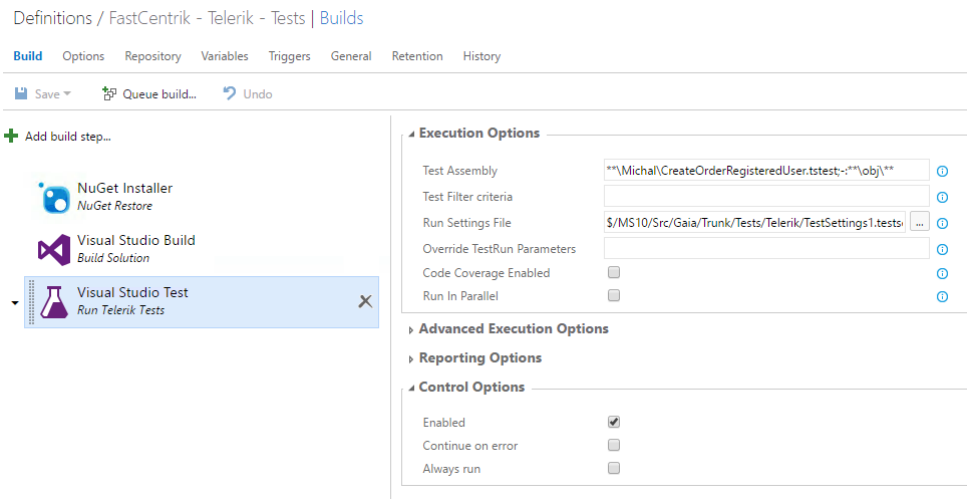
TFS je nástroj od spoločnosti Microsoft, ktorý umožňuje spravovať zdrojový kód, robiť záznamy, spracovať požiadavky, testovať a automatizovať zostavenie systému. Umožňuje spravovať agilné procesy, sledovať prácu a určovať pracovné priority.

TFS je dostupné v dvoch rôznych formách, v priestoroch alebo ako online. Druhá forma sa nazýva Visual Studio Team Services. Cloudová služba je podporovaná platformou Microsoft Azure. Používa rovnaký kód ako prvá forma TFS, s drobnými úpravami a implementuje najnovšie funkcie. Visual Studio Team Services nevyžaduje žiadnu inštaláciu. Užívateľ sa jednoducho prihlási cez Microsoft konto, vytvorí projekty a pridá členov tímu. Nové funkcie, ktoré sú vyvinuté v krátkych časových cykloch sú implementované do online služby ako prvé.[12]

9.2 Automatizované testy v procese zostavenia

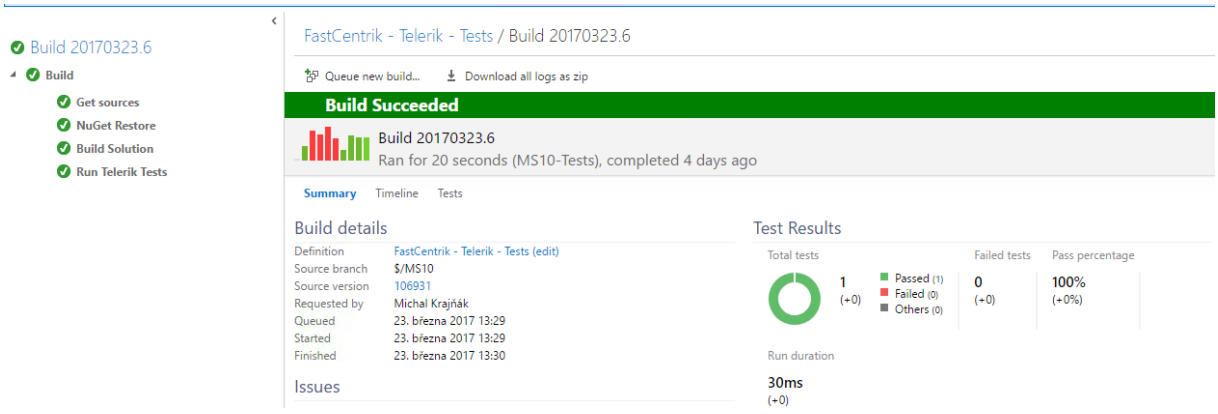
Pre zavedenie automatizovaných testov do procesu zostavenia je potrebné mať nainštalovaný nástroj Telerik Testing Framework. Ďalej je potrebné mať na počítači nainštalovaného zostavovacieho agenta. Po nainštalovaní týchto častí môžeme prejsť k samotnému nastaveniu potrebných vecí pre zavedenie testov do procesu zostavenia. Na obr. 13 môžeme vidieť nastavenie parametrov, ktoré sú potrebné, aby proces zostavenia zahrnul do seba automatizované testy. Parameter **Test Assembly** nám umožňuje nastaviť konkrétny test, ktorý chceme zaviesť. Ak by sme chceli zaviesť, aby sa všetky testy spracovali pri zostavení, tak môžeme napísať do parametra Test Assembly hodnotu **"*åii"**. Ďalší dôležitý parameter, ktorý je potrebné nastaviť je **Run Settings File**.Tento parameter umožňuje načítať nastavenia, ktoré sú špecifické pre

daný testovací projekt. Sú v ňom veci ako štandardný webový prehliadač, bázová URL adresa, simulácia vstupov atď.



Obrázok 13: Nastavenie parametrov procesu zostavenia

Po nastavení jednotlivých hodnôt môžeme náš proces zostavenia spustiť. Po spustení sa stiahnu potrebné Nuget balíčky, zostaví sa projekt a spusti sa automatizovaný test. Máme možnosť pridať ďalšie kroky k nášmu procesu, ako je napr. nasadenie na server alebo na cloudovú službu Microsoft Azure. Na obr. 14 môžeme vidieť výsledok celkového procesu zostavenia. Jediná nevýhoda, ktorá spočíva v tomto zostavovaní je, že ak máme automatizované testy, pri ktorých využívame simuláciu reálneho kliknutia alebo vkladania vstupu, tak musíme mať náš počítač neustále zapnutý, byť prihlásený a mať vypnutý šetrič obrazovky. Bohužiaľ to predstavuje určité bezpečnostné riziko pre spoločnosť, jedine, že by sa daný počítač zamkol do nejakej miestnosti a tým zvýšil bezpečnosť.



Obrázok 14: Celkový výsledok procesu zostavenia

10 Automatizované a manuálne testovanie z hľadiska času

Pri samotnom manuálnom testovaní strácame nielen peniaze, ale hlavne aj čas. V tejto kapitole si uvedieme experimentálne príklady, pri ktorých dokážeme, že automatizované testovanie je z hľadiska tvorby, vykonávania a údržby časovo výhodnejšie, než manuálne testovanie.

Každá tvorba automatizovaného testu a manuálneho testu sa skladá z určitých krokov. Jednotlivé kroky pre testera zaberajú určitý čas. V nasledujúcom zozname si uvedieme kroky tvorby, ktoré sú potrebné pre automatizované testovanie:

1. Analýza funkčnosti.
2. Vytvorenie testovacieho prípadu.
3. Vytvorenie automatizovaného testu.
4. Spustenie vytvoreného automatizovaného testu.
5. Údržba automatizovaného testu.

Manuálne testovanie, ako vieme z úvodu práce, je menej zložitejší proces a vyžaduje si aj menej kvalifikovaného testera pre jeho vykonanie. Z toho dôvodu je potrebných menej krokov pri tvorbe, než u automatizovaného testovania. V nasledujúcom zozname sú kroky tvorby pre manuálne testovanie:

1. Analýza funkčnosti aplikácie.
2. Vytvorenie testovacieho prípadu.
3. Ručné vykonania manuálneho testu podľa testovacieho prípadu.

Vidíme, že automatizované testovanie zahŕňa viac krokov ako manuálne testovanie. Je to dané tým, že automatizované testovanie je zložitejší proces a najviac času pri jeho tvorbe zaberá samotná implementácia testovacieho skriptu. V obidvoch prípadoch treba na začiatku testovania analyzovať funkčnosť, ktorú ideme testovať. Najskôr je potrebné oboznámiť sa s funkčnosťou, ktorú ideme testovať a až potom môžeme vykonávať testovanie. Pretože, ak zle pochopíme, napr. ako funguje objednávkový proces, tak nemôžeme ani začať s písaním testovacieho prípadu alebo samotného skriptu.

Ďalej vidíme, že v obidvoch prípadoch vytvárame testovací prípad. Je dobrou radou najskôr vytvoriť testovací prípad pred samotnou implementáciou testovacieho skriptu.

Pri automatizovanom testovaní máme ako ďalší krok tvorbu samotného testovacieho skriptu. Tento krok zaberie v celom automatizovanom testovaní najviac času, pretože samotné vytvorenie dobrého znovupožitelného testu si vyžaduje určitý čas na dobrý návrh a implementáciu testu. U manuálneho testu tento krok nemusíme riešiť, pretože pristupujeme rovno k vykonaniu testu podľa testovacieho prípadu.

Ako posledné kroky pri automatizovanom testovaní sú kroky spustenia testov a údržba. Každý vytvorený automatizovaný test sa bude určite v budúcnosti musieť aspoň trochu upraviť, pretože vývojári softvérovej aplikácie robia zmeny a je možné, že táto zmena môže zasiahnuť samotný test. Tu je rozdiel oproti manuálnemu testovaniu, pretože manuálne testovanie zahŕňa ako posledný krok len samotné vykonávanie testov. Netreba teda udržiavať manuálne testy, maximálne tak udržiavať testovací prípad testu.

Všetky tieto kroky, ako pre automatizované, tak aj pre manuálne testovanie si v nasledujúcich podkapitolách analyzujeme na konkrétnych testov a následne zhodnotíme aj celkové porovnanie z hľadiska času.

10.1 Analýza manuálneho a automatizovaného testovania

V tejto podkapitole si preberieme jednotlivé kroky tvorby pre manuálne a automatizované testovanie. Ako sme si hovorili v úvode kapitoly, tak automatizované testovanie zaberá viac krokov pri tvorbe ako manuálne testovanie. To však neznamená, že výsledný efekt bude zdĺhavejší. Niekedy sa oplatí investovať čas do zložitejšej tvorby a výsledné používanie sa ukáže ako efektívne. To si teraz dokážeme prostredníctvom ukážkových testov z predchádzajúcej podkapitoly.

10.1.1 Oboznámenie sa s testovanou funkčnosťou

Ako sme už spomínali, najskôr je nutné testovanú funkčnosť analyzovať a oboznámiť sa s ňou a až potom je možné ju testovať. Keď si zoberieme napr. testy na registrovanie zákazníka, objednávanie neregistrovaným zákazníkom a prihlásenie užívateľa, tak pri oboznamovaní s týmito funkčnosťami nie je nejaký veľký rozdiel, ak sa jedná o automatizované alebo manuálne testovanie. Tento krok vykonáva človek a nie stroj, tak záleží od človeku, ako rýchlo pochopí danú funkčnosť.

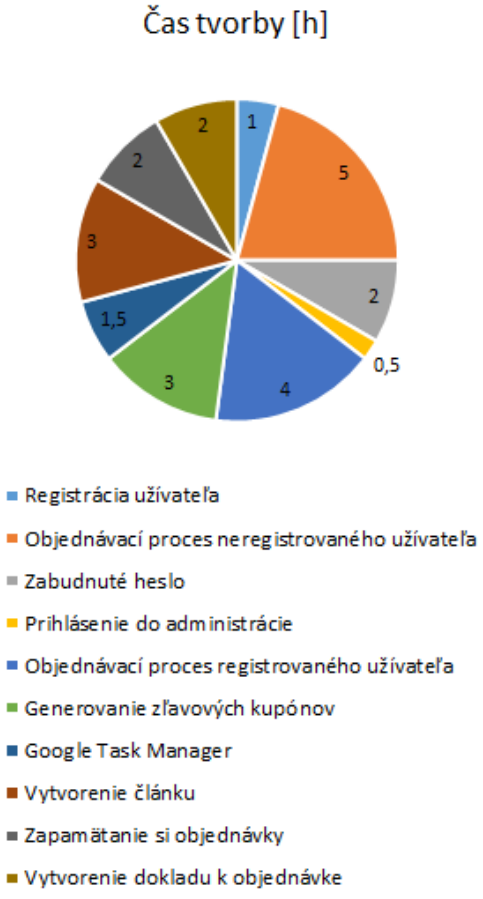
10.1.2 Tvorba testovacieho prípadu

Ako ďalší krok, po oboznámení sa s funkčnosťou aplikácie, je vytvorenie testovacieho prípadu. Toto vytvorenie platí rovnako ako pre manuálne, tak aj pre automatizované testovanie. Dĺžka vytvorenia tohto testovacieho prípadu je daná veľkosťou a zložitosťou samotnej testovanej funkcionality. Ak ide, napr. o testovací prípad prihlásenia, tak tam máme len pár krokov, ale ak sa jedná už o objednávací proces, tak sa nám tá tvorba predĺži väčším množstvom krokov a dosadením vstupných dát.

Pri automatizovanom testovaní to môže byť o niečo dlhšie, pretože pridávame pár nových atribútov, ako sme videli vyššie v kapitolách. Tieto atribúty nám väčšinou charakterizujú cestu k testovaciemu skriptu, ku konkrétnym testovacím dátam v XML, SQL alebo v inom dátovom súbore. V tomto prípade nemusíme opäť predpokladať nejaký veľký časový rozdiel v oboch typoch testovania.

10.1.3 Tvorba testovacieho skriptu

Tento krok tvorby sa týka len automatizovaného testovania, pretože u manuálneho testovania nám postačí už priamo testovací prípad na testovanie. Testovací skript je najťažšou a najviac časovo náročnou časťou u samotného automatizovaného testovania. Tiež záleží od testovacieho nástroja, čo všetko nám môže pri tvorbe zjednodušiť. Čo všetko nám umožňuje vygenerovať a čo všetko si musíme urobiť sami. Samotná implementácia testu, ktorá od testera, ktorý ju vykonáva vyžaduje určité znalosti programovania a znalosti konkrétneho programovacieho jazyka, ktoré testovací nástroj podporuje. Orientačnú dobu vytvorenia testovacieho skriptu pre vybrané testované funkčnosti systému FastCentrik môžeme vidieť na obr. 15.

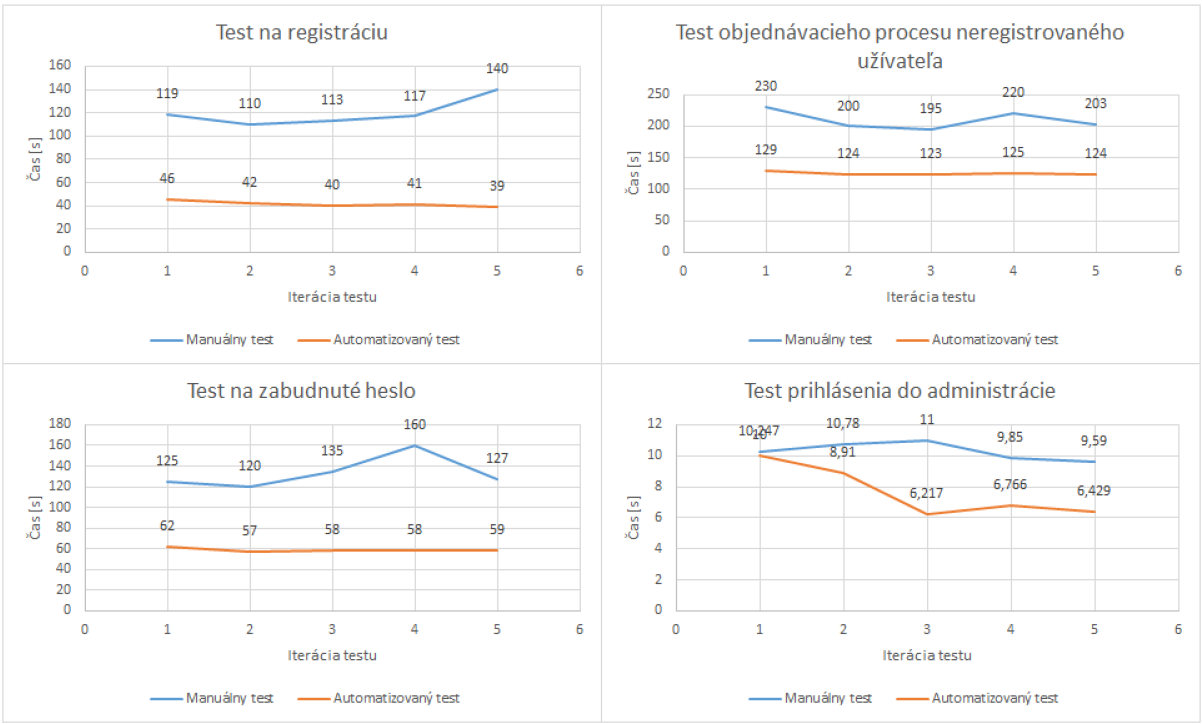


Obrázok 15: Graf zobrazujúci čas tvorby testovacích skriptov

Z uvedeného grafu vidíme, že objednávkový proces zaberá najviac času, pretože zahŕňa viac krokov, viac komplexných vecí a pracuje ako na užívateľskej časti, tak aj na administračnej časti. Najmenej času zaberá prihlásenie užívateľa, pretože tam nemáme veľa logiky a stačí nám vedieť dobre validovať prihlasovacie meno a heslo.

10.1.4 Spustenie samotného testu

Ak sme s funkčnosťou, ktorú ideme testovať oboznámení, ak máme vytvorené testovacie prípady a ak máme vytvorený testovací skript pre automatizované testovanie, tak môžeme začať so samotným vykonávaním testovania. V nasledujúcich grafoch si ukážeme časové porovnanie manuálneho a automatizovaného vykonávania testov v závislosti na počte vykonávaných iterácií testu. Iterácia testu znamená počet spustení testu. Dáta, z ktorých boli vytvorené grafy, boli získané z reálnych vykonávaných experimentov v spoločnosti NetDirect, s.r.o. Na obr. 16 vidíme štyri grafy, ktoré znázorňujú štyri vybrané testované funkčnosti a porovnanie vykonávania testu medzi manuálnym a automatizovaným testovaním. Čas získaný pre automatizované testovanie bol získaný z výsledku dokončenia testov v nástroji Telerik Test Studio. Čo sa týka manuálneho testovania, tak tento čas bol získaný z vykonávania dvoch testeriek zo spoločnosti NetDirect, s.r.o a to formou zaznamenávania cez stopky.



Obrázok 16: Grafy ukazujúce porovnanie manuálnych a automatizovaných testov

Na tomto grafe vidíme, že testy manuálnym testovaním trvajú dvakrát dlhšie než automatizovaným testovaním. Je to dané viacerými faktormi, ktoré to ovplyvňujú. Testovací skript pre automatizované testovanie je naprogramovaný, sú v ňom nastavené testovacie dáta a vie vždy presne, čo má vykonať. Naproti tomu manuálny test vykonáva človek, ktorý musí ručne zadávať dáta, musí sa zorientovať na každej stránke a môže sa stať, že aj urobí chybu.

Na grafe môžeme vidieť, že krivka u manuálneho testovania má väčšie odchýlky, než u automatizovaného testovania. To je tým, že u manuálneho testera nemôžeme predpokladať, že každý

test vykoná za ten istý čas. Môžu ho ovplyvniť rôzne veci, ako napr. urobenie chyby, môže na niečo zabudnúť alebo môže byť pri poslednej iterácií unavený a môže mu to ísť pomalšie. U automatizovaného testovania je zase krivka skoro rovnomerná. Pretože počítač vykonáva tento typ testovanie, tak mu nič nebráni v tom, aby aj pri poslednej iterácií nepodal taký istý výsledok ako pri prvej iterácií.

Môžeme si všimnúť, že pri všetkých testoch, okrem testu na prihlásenie do administrácie, sa nám čas skrátil o raz taký čas, ako pri manuálnom testovaní. Pri prihlásení do administrácie sa jedná o test celkom jednoduchý, a preto jeho vykonanie zaberá krátky čas a aj rozdiel oproti manuálnemu testovaniu nie je až taký veľký. Ide len o pár sekúnd, avšak, ak používame prihlásenie pri väčšine iných testov, tak to už má značný vplyv.

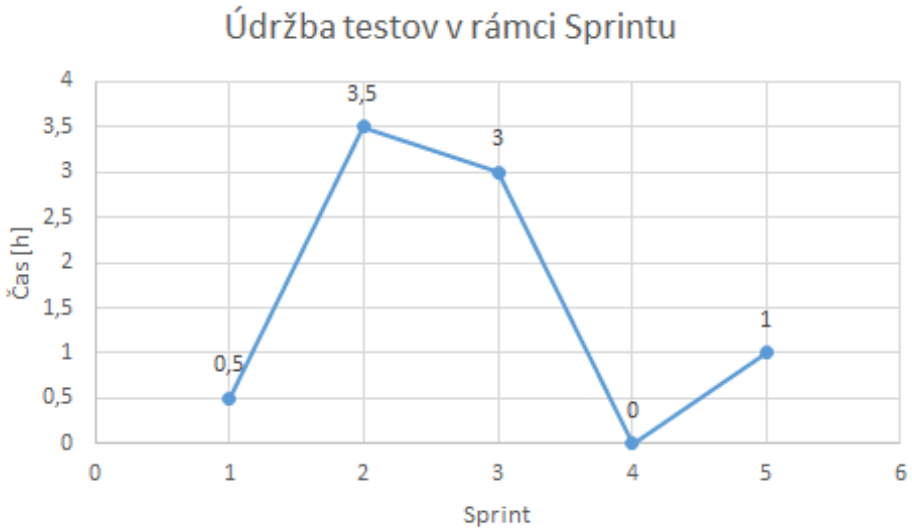
Zaujímavé, čo si môžeme z grafov všimnúť, je, že pri automatizovanom testovaní vždy pri druhej iterácii je čas vykonania testu kratší. Je to tým, že pri prvej iterácii sa musí spustiť nová inštancia webového prehliadača, musia sa spustiť potrebné procesy pre test a až potom prebieha samotné testovanie. Pri ďalších iteráciách sa už ale vychádza zo spustenej inštancie prehliadača, a preto je už vykonávanie testu o niečo rýchlejšie.

10.1.5 Údržba testov

Každý vytvorený testovací skript pre automatizované testovanie bude nutné niekedy v budúcnosti aspoň trochu zmeniť. Je veľmi nepravdepodobné, že naše testovacie skripty sa ani raz nebudú musieť upravovať. U manuálneho testovania tento problém nehrozí, pretože tam nemáme žiadne testovacie skripty a maximálne sa budú upravovať a udržiavať testovacie prípady.

Najväčšia údržba postihuje testy, ktoré sú spojené s častou zmenou grafického rozhrania. Keď si zoberieme ako konkrétny príklad test na vytvorenie horného menu, tak tento test prebieha na užívateľskej časti systému FastCentrik a táto časť poskytuje pre zákazníkov rôzne šablóny. Rôzne šablóny však prinášajú aj rôzne problémy pri testovaní, pretože prebiehajú častými grafickými zmenami.

V rámci iterácií vývoja bolo nutné vykonávať opravy na automatizovaných testoch. Na obr. 17 vidíme graf, ktorý znázorňuje celkový čas v hodinách, ktorý bol potrebný na opravu testov v rámci iterácií Sprintu.



Obrázok 17: Graf znázorňujúci čas potrebný na opravu testov

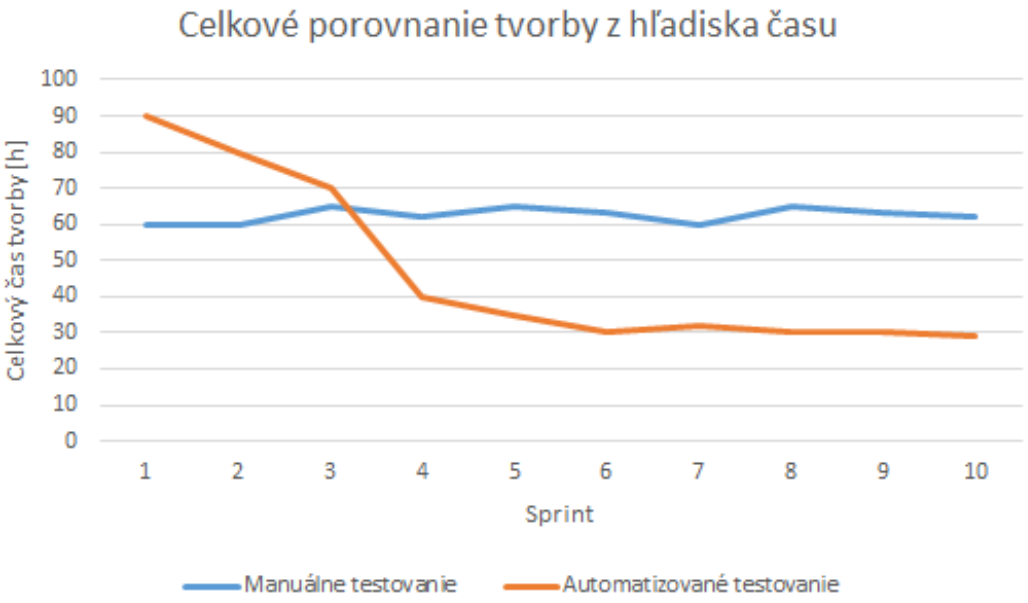
Na tomto grafe vidíme, že krivka nie je rovnomerná a je rozdielna v závislosti na danej iterácii sprintu. V nasledujúcom zozname si popíšeme jednotlivé iterácie a dôvod, vďaka ktorému je krivka grafu taká nerovnomerná:

- 1. iterácia - v tejto iterácií sa jednalo o chybu, ktorú zahlásil klient, preto bolo nutné aj opraviť určité testy.
- 2. iterácia - na tejto iterácií vidíme, že zaberá najviac času spomedzi všetkých ostatných iterácií. Je to preto, lebo testy museli byť opravené z dôvodu upravovania šablóny. Zmeny v šablóne majú najväčší dopad na automatizované testy.
- 3. iterácia - boli pridané nové funkcionality do procesu objednávky a tiež boli vykonané nejaké zmeny v objednávke. Preto všetky testy, ktoré pracovali s objednávkou museli byť opravené.
- 4. iterácia - v tejto iterácií vidíme, že neobsahuje žiadny čas. Je to preto, lebo neboli pridané žiadne zmeny alebo definovali sa nové funkcionality s prioritou 3 a 4. Pre tieto menej prioritné funkcionality sa po dohode s tímom nevytvárali automatizované testy.
- 5. iterácia - vykonávali sa drobné úpravy v testoch na užívateľský účet.

Ako môžeme vidieť z uvedeného zoznamu iterácií, tak všetko závisí od toho, aké funkcionality sa budú meniť v priebehu iterácie. Ak ide o rozsiahle zmeny v grafickom rozhraní, tak môžeme očakávať, že ten časový nárast bude väčší.

10.2 Zhodnotenie porovnania

Na základe predošlých porovnaní v rámci stanovených krokov tvorby je zrejmé, že počiatočný čas investovaný do automatizovaného testovania je vyšší než u manuálneho testovania. V rámci počiatočného automatizovaného testovania je najťažšie oboznámiť sa so samotným testovacím prostredím a implementovať testovacie skripty. Manuálne testovanie je z hľadiska času rovnaké pri každej iterácii systému. Na obr. 18 môžeme vidieť graf, ktorý znázorňuje celkový čas investovaný do testovania v rámci celého životného cyklu.



Obrázok 18: Graf znázorňujúci porovnanie celkového času manuálneho a automatizovaného testovania

V nasledujúcom zozname si popíšeme manuálne a automatizované testovanie, na základe ktorého boli získané časové údaje pre graf:

- Manuálne testovanie - tento typ testovania vykonávali dve testerky zo spoločnosti NetDirect, s.r.o. Prácu testovania si z časti rozdelili a dôležité časti kontrolovali obidve. Čas pre celkové spustenie manuálnych testov si testerky zaznamenávali cez stopky.

- Automatizované testovanie - časy pre vykonávanie testov sme získali z výsledného záznamu, ktorý nám poskytol nástroj Telerik Test Studio. Celková doba vytvorenia testovacích skriptov bola získaná z verzovacieho nástroja TFS. Každý jeden testovací skript mal priradený čas na vytvorenie a tým sme mohli túto dobu zaznamenať.

Na grafe môžeme vidieť, že automatizované testovanie vyžaduje pri začiatku tvorby podstatne vyšší čas, než manuálne testovanie. Je to dané tým, že prvotné oboznámenie s testovacím nástrojom, testovacími skriptami vyžaduje viac času. Postupom času už od tretej iterácie vidíme,

že automatizované testovanie má kratší čas, než manuálne testovanie. Ako náhle už máme vytvorené testovacie skripty pre naše kritické funkcionality, tak potom už len stačí naše testovacie skripty udržiavať alebo niekedy vytvoriť nový testovací skript, ak by sa jednalo o novú kritickú funkcionalitu.

10.3 Odporúčania pre zlepšenie času automatizovaných testov

Z predchádzajúcich podkapitol sme zistili, že automatizované testovanie je oveľa rýchlejšie ako manuálne testovanie. Môžeme ale jeho rýchlosť ešte viac zvýšiť? Áno, môžeme, avšak za určitých podmienok, ktoré budeme pri vývoji automatizovaných testov dodržiavať.

10.3.1 Jednoduchosť testu

V rámci dobrej rady ako dosiahnuť ešte kratší čas pri automatizovaných testov je, že budeme vytvárať testy, ktoré sú jednoduché. Síce máme prostriedky a technológie k tomu, aby sme boli schopní vytvoriť aj komplexný test, avšak odzrkadlilo by sa to na jeho časovej úprave.

Ak už máme vytvoriť komplexný test, tak si ho môžeme rozdeliť na viac jednoduchších testov a tým to bude opäť časovo menej náročnejšie na údržbu.

10.3.2 Dobré vyhľadávacie metódy

Ďalšou radou ako dosiahnuť lepší výsledný čas je používať správne vyhľadávacie metódy. Síce nám nástroj Telerik Test Studio ponúka množstvo vyhľadávacích metód, treba ale vedieť, ktoré metódy nám zaručia väčšiu znovupoužiteľnosť a menej častú úpravu testu.

Zlý výber vyhľadávacích metód môže mať za následok veľké množstvo pádu testov. Preto musíme pri výbere postupovať rozumným spôsobom. Možno nám správny výber metódy zaberie viac času, ale nemusíme sa potom trápiť s úpravou testu. Dobré metódy pre vyhľadávanie môžu byť, napr. metódy na hľadanie podľa ID elementu alebo aj metóda, ktorá hľadá podľa názvu HTML elementu.

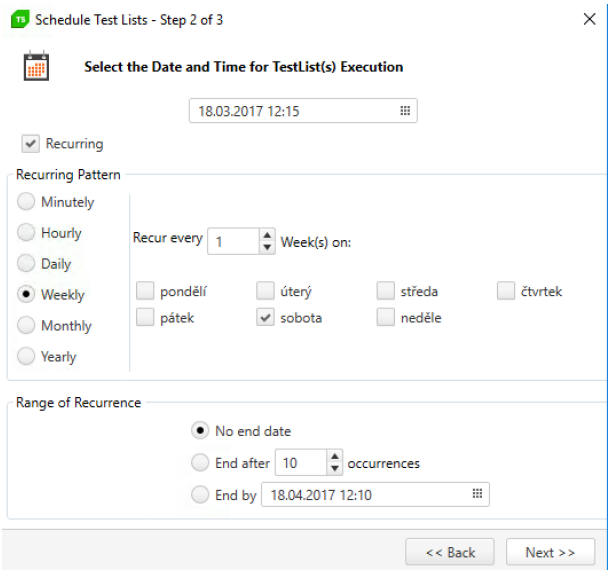
10.3.3 Tvorba pomocných tried

V kapitole, kde sme preberali tvorbu automatizovaných testov pre systém FastCentrik, tak sme si hovorili niečo aj o pomocných triedach, ktoré sme pri práci využili. Je tiež veľkou úsporou času, ak si vytvoríme vlastné pomocné triedy, ktoré nám budú udržiavať určitú logiku funkcionality. Zabránime tým duplicitnému kódu a tiež znížime čas potrebný na úpravu testu. Stačí nám v jednej funkcii niečo zmeniť a prejaví sa to vo všetkych ostatných častiach, kde je funkcia použitá.

10.3.4 Plánované spúšťanie testov

Telerik Test Studio nám umožňuje vybrať nejaký testovací zoznam a spustiť ho v nejakom stanovenom čase. Z nášho lokálneho počítača si môžeme nastaviť, aby plánované spúšťanie testov prebiehalo buď na ostatných počítačoch v sieti alebo na virtuálnych strojoch. Ak máme veľké množstvo testov, tak si ich môžeme rozdeliť medzi viaceré počítače a tým sa podstatne zredukuje celkový čas vykonávania testov.

Ďalšia vec, ktorá je tu výhodná je, že si môžeme toto plánovanie nastaviť, napr. cez noc a ráno, keď prídeme do práce si môžeme pozrieť výsledky všetkých testov. Nie sme teda obmedzení pracovnou dobou testerov, ako to je u manuálneho testovania. Na obr. 19 môžeme vidieť ukážku nastavenia plánovaného spustenia testov v nástroji Telerik Test Studio.



Obrázok 19: Nastavenie plánovaného spustenia automatizovaných testov

10.3.5 Odstraňovanie vytvorených položiek

Ďalšou tiež dôležitou radou, ktorá nám zjednodušuje testovanie a skracuje celkový čas je odstraňovanie položiek, ktoré boli vytvorené automatizovanými testami. Ak používame testy, v ktorých vytvárame, napr. užívateľa s jedinečným prihlasovacím menom a v danom teste nerieši me jeho odstránenie na konci, tak nám to môže veľmi skomplikovať celý proces. Toto odstránenie položiek síce môže byť zavedené v predpokladoch testu, avšak človek môže na ne zabudnúť a tým, ak by sme spustili test, tak by zbytočne zlyhal. Ak by sa jednalo o veľmi zložitý a dlhší test, tak by nás to veľmi mrzelo, pretože by sme prišli o drahocenný čas.

Tiež pre testera, ktorý spúšťa automatizované testy a videl by v predpokladoch napísané, že sa majú tie a tie položky vymazať, tak by to bolo veľmi zdĺhavé manuálne odstraňovať tieto veci. Preto je lepšie v samotných testoch naimplementovať toto odstránenie na konci každého testu,

aby stav databázy daného systému bol rovnaký ako pred spustením testu. Tým zabránime aj problémom ostatných testerov, ktorí vykonávajú testovanie nad rovnakým systémom.

11 Záver

Hlavným cieľom tejto diplomovej práce bolo vykonať empirický výskum, ktorý overí funkčnosť systému FastCentrik s ohľadom na časovú efektivitu za použitia manuálnych a automatizovaných testov. Na základe daného výskumu môžeme stanoviť, že čas investovaný do automatizovaných testov má zmysel a náklady, ktoré boli nutné pre zavedenie testov sa nám časom vrátia, ba dokonca spoločnosť bude viac zisková.

Na začiatku práce sme začali s porovnávaním testovacích nástrojov, kde sme zistili, ktorý nástroj má aké výhody. Na základe daného porovnania a po konzultácii v spoločnosti sme sa rozhodli použiť nástroj Telerik Test Studio. Následne sme začali s tvorbou testovacích scenárov, testovacích prípadov a testovacích skriptov. Tieto automatizované testy boli vytvorené pre užívateľskú a administračnú časť systému.

Následne sme jednotlivé časti systému rozdelili podľa kritickosti a stanovili, ktoré funkčnosti systému sú pre nás najdôležitejšie. Pre tieto najviac kritické časti systému boli vytvorené automatizované testy. Zahŕňali ako administračnú časť systému, tak aj užívateľskú časť systému. Pre detailnejšie znázornenie pokrytých častí systému automatizovanými testami sme vytvorili mapu, ktorá bude do budúcnosti slúžiť pre ostatných testerov vo firme. Všetky vytvorené testovacie scenáre, testovacie prípady, testovacie skripty a mapa pokrytia sú obsiahnuté v priloženom CD.

Pre to, aby celý proces zavedenia automatizovaných testov bol efektívny, sme tieto testy zaviedli do samotného procesu zostavenia. Tým sa zredukuje počet chýb a zamedzíme tomu, aby sa tieto chyby vyskytli, keď už bude systém nasadený na servery. V rámci vytvorenia automatizovaných testov sme sa stretli s rôznymi problémami, ktoré nám bránili v tom, aby tvorba testovacích skriptov prebiehala jednoducho, rýchlo a efektívne. Preto sme v práci popísali aj rôzne problémy, ktoré sa vyskytli v samotnom systéme FastCentrik a k nim jednotlivé odporučenia, ktoré nám umožňujú tieto problémy vyriešiť.

V závere práce sme vykonávali analýzu medzi manuálnym a automatizovaným testovaním. V tejto analýze sme pracovali s celkovým životným cyklom testov od oboznámenia sa s funkčnosťou až po údržbu testov. Táto analýza zahrňovala reálne dáta, ktoré sme získali na reálnych experimentoch v spoločnosti NetDirect, s.r.o. Podstatou tejto práce je, aby slúžila aj ako návod pre ostatných testerov, ktorí budú využívať vytvorené automatizované testy a aby nemali problémy či už s tvorbou nových testov alebo s údržbou existujúcich testov.

Pri tejto práci som využil znalosti, ktoré som získal z predmetu **Testovanie a softvérová kvalita**. Prínosom v rámci vykonávania diplomovej práce bolo, že som mohol byť súčasťou pri prvotnom zavedení automatizovaných testov do spoločnosti NetDirect, s.r.o. Mohol som byť prítomný v rámci celého životného cyklu automatizovaných testov od vytvorenia testovacích prípadov až po zavedenie do procesu zostavenia. Ďalším prínosom pre mňa bolo, že som si prakticky mohol vyskúšať vytvorenie testovacích skrípt na základe testovacích prípadov. Mohol som si teda rozšíriť svoje odborné znalosti o testovacie znalosti, ktoré si myslím, že majú veľmi veľký význam pri vývoji akéhokoľvek softvérového produktu.

Literatúra

[1] ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: průvodce testováním [online]. Brno: Computer Press, 2013 [cit. 2017-01-23]. ISBN 978-80-251-3816-8.

[2] Testování softwaru [online]. [cit. 2017-01-10]. Dostupné z: http://www.testovanisoftwaru.cz

[3] FINK, Mark. The Hitchhiker's Guide to Test Automation: Software Quality Assurance and Test Automation in Practice. Great Britain: Amazon.co.uk, 2013. ISBN 9781456486129.

[4] MYERS, Glenford J., Corey SANDLER a Tom. BADGETT. The art of software testing [online]. 3rd ed. Hoboken, N.J.: John Wiley, c2012 [cit. 2017-01-23]. ISBN 11-181-3314-5.

[5] Software Testing Tutorial [online]. [cit. 2017-01-14]. Dostupné z: https://www.tutorialspoint.com/software_testing

[6] Scrum [online]. [cit. 2017-01-14]. Dostupné z: http://scrum.sk/o-agile-a-scrum/scrum-agilny-projektovy-manazment/

[7] Test Studio Overview [online]. [cit. 2017-01-15]. Dostupné z: http://www.docs.telerik.com/teststudio/

[8] Selenium - Web Browser Automation [online]. [cit. 2017-01-24]. Dostupné z: http://www.seleniumhq.org/

[9] Automated Software Testing Made Simple | Test Complete [online]. [cit. 2017-01-24]. Dostupné z: https://smartbear.com/product/testcomplete/overview/

[10] Tvorba a pronájem e-shopů, tvorba www stránek | NetDirect [online]. [cit. 2017-01-31]. Dostupné z: http://www.netdirect.cz/

[11] MCDA: Multi-Criteria Decision Analysis [online]. [cit. 2017-03-17]. Dostupné z: https://www.ncsu.edu/nrli/decision-making/MCDA.php

[12] Team Foundation Server [online]. [cit. 2017-03-30]. Dostupné z: https://www.visualstudio.com/tfs/

A Obsah priloženého CD

1. diplomovaPraca.pdf - text diplomovej práce,
2. Telerik - zdrojový kód celého testovacieho projektu Telerik Test Studio,
3. testovaciePripady.docx - celkové testovacie scénare a testovacie prípady systému FastCentrik.

B Zoznam vytvorených testovacích scenárov pre systém Fast-Centrik

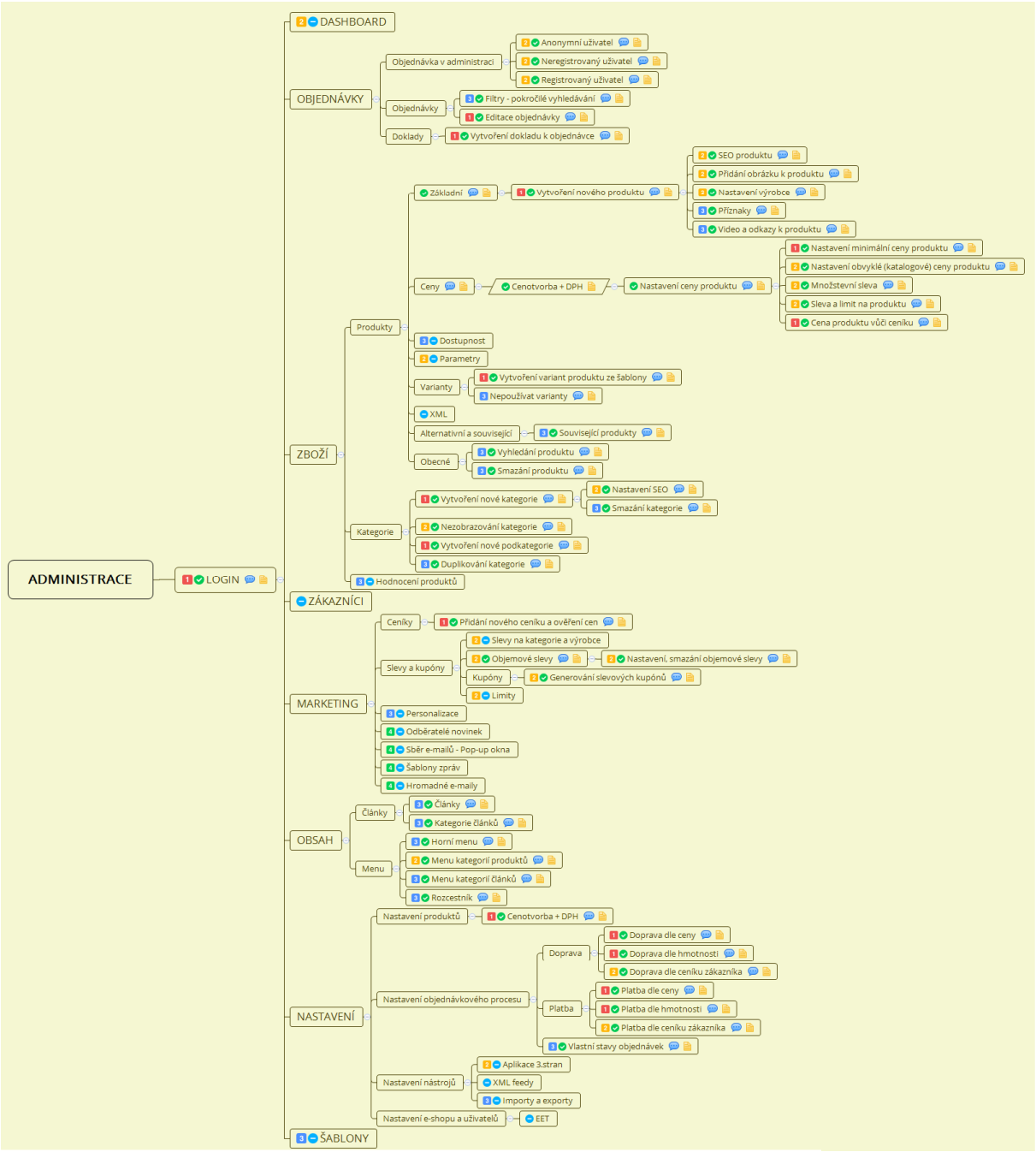
Pre administračnú časť sú v nasledujúcom zozname uvedené funkcionality, pre ktoré boli vytvorené automatizované testy:

1. Funkčnosť dopravy v závislosti na cene objednávky
2. Funkčnosť dopravy v závislosti na hmotnosti objednávky
3. Funkčnosť dopravy v závislosti na cenníku užívateľa
4. Generovanie zľavových kupónov
5. Vytvorenie dokladu k objednávke
6. Prihlásenie užívateľa do administrácie
7. Filtrovanie objednávky
8. Tvorba objednávky pre neregistrovaného užívateľa
9. Tvorba objednávky pre registrovaného užívateľa
10. Tvorba objednávky pre anonymného užívateľa
11. Editácia vytvorenej objednávky
12. Vytvorenie a použitie vlastného stavu objednávky
13. Funkčnosť platby v závislosti na cene objednávky
14. Funkčnosť platby v závislosti na hmotnosti objednávky
15. Funkčnosť platby v závislosti na cenníku užívateľa
16. Vytvorenie článkov
17. Vytvorenie kategórie článkov
18. Vytvorenie horného menu
19. Vytvorenie menu pre kategóriu produktov
20. Vytvorenie menu pre kategóriu článkov
21. Vytvorenie menu rozdeľovač

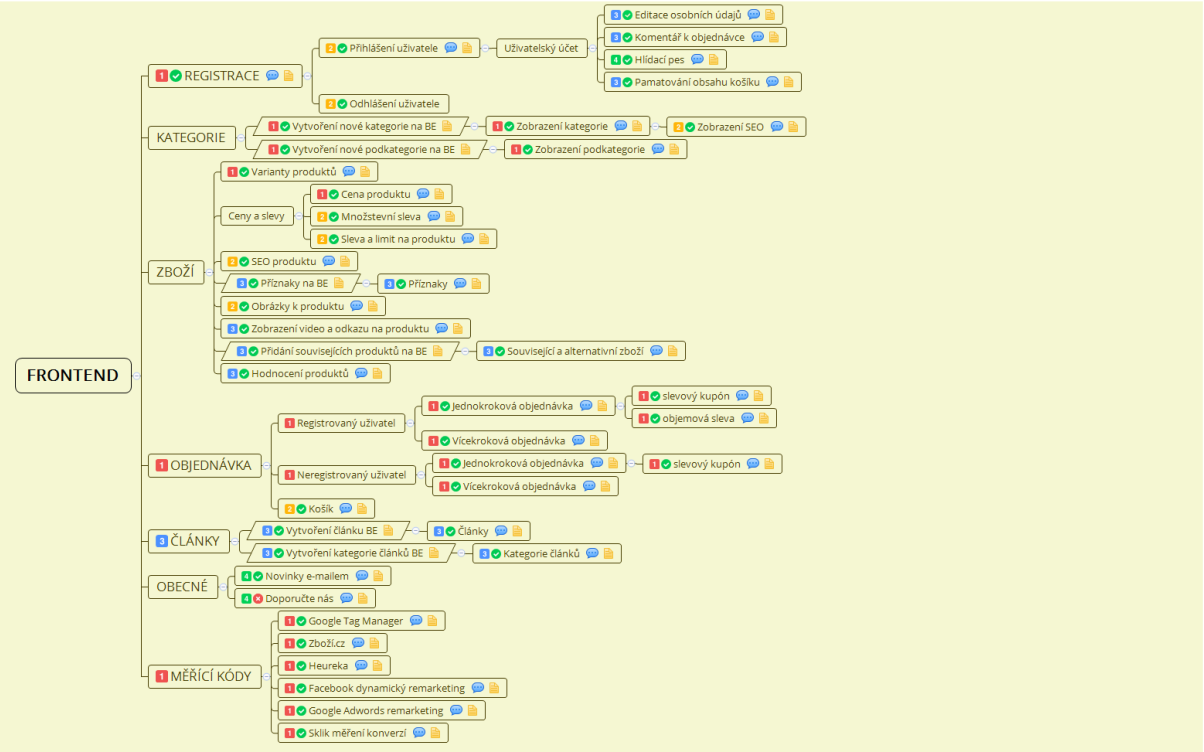
Pre užívateľskú časť sú v nasledujúcom zozname uvedené funkcionality, pre ktoré boli vytvorené automatizované testy:

1. Editácia užívateľského účtu
2. Funkčnosť zabudnutého hesla
3. Prihlásenie do užívateľského účtu
4. Registrácia zákazníka
5. Funkčnosť strážneho psa pre produkty
6. Hodnotenie produktov
7. Funkčnosť zapamätania si objednávky
8. Funkčnosť objednávania produktov v jednokrokovej objednávke
9. Funkčnosť objednávania produktov vo viackrokovej objednávke
10. Zasielanie noviniek e-mailom
11. Funkčnosť posielania článkov známym
12. Funkčnosť posielania správ k objednávke
13. Funkčnosť správneho zobrazenia horného menu
14. Funkčnosť správneho zobrazenia menu pre kategóriu produktov
15. Funkčnosť správneho zobrazenia menu pre kategóriu článkov
16. Funkčnosť správneho zobrazenia menu rozdeľovač
17. Funkčnosť zobrazenia článkov
18. Merací kód Google Adwords
19. Merací kód Google Tag Manager
20. Merací kód Facebook
21. Merací kód Heureka
22. Merací kód Sklik
23. Merací kód Zbozi.cz

C Mapa pokrytia pre administračnú časť systému FastCentrik



D Mapa pokrytia pre užívateľskú časť systému FastCentrik



E Legenda pre mapu pokrytia

 - Tento symbol u testu vyjadruje pre testera, že test prebehol v poriadku a neobsahuje žiadne chyby

 - Tento symbol vyjadruje, že pre danú funkčnosť zatiaľ ešte neexistuje automatizovaný test

 - Tento symbol u funkčnosti obsahuje menu testera, ktorý daný test vytvoril. Tieto informácie je možné zobraziť iba v nástroji XMind.

 - Tento symbol u funkčnosti obsahuje informácie o tom, aké dáta test obsahuje a nejaké dodatočné informácie. Opäť sú tieto informácie dostupné len v nástroji XMind.

 - Tento symbol u testu vyjadruje, že test neprebehol v poriadku a chyba je zrejme na strane testera

 - Priorita 1

 - Priorita 2

 - Priorita 3

 - Priorita 4

F Ukážka testovacieho scenára pre užívateľskú časť systému

- **ID**: 10
- **Popis**
  - Test umožňuje skontrolovať, či objednávanie produktov registrovaným zákazníkom prebehlo úspešne
- **Dáta**
  - order/createOrderOnFrontend/registeredUsersOnFrontend/registeredUserOnFrontend
  - login/frontend/successfull/userLoginFrontendSuccessful
- **Testovací skript**
  - FE/Ordering FE/OrderingProduct/With registered user/OrderingProductRegisteredUser.tstest
- **Predpoklady**
  - Nie sú kladené žiadne predpoklady
- **Kroky**
  1. Vykonáme test na prihlásenie do administrácie s id: 1
  2. Prejdeme do záložky **Objednávkový proces**
  3. Vyberieme typ objednávky: **Data[“oneStepOrder“]**
  4. Klikneme na tlačidlo **Uložit**
     - Očakávaný výsledok
       * Bola zobrazená úspešná správa o uložení
  5. Vykonáme test na prihlásenie do užívateľského účtu s id: 3
     - Očakávaný výsledok
       * Užívateľ je úspešne prihlásený a vidí tlačidlo na odhlásenie
  6. Užívateľ prejde na produkt na základe jeho URL adresy: **Data[“productUrl“]**
     - Očakávaný výsledok
       * Užívateľ vidí zobrazený produkt na stránke
       * Je správne spočítaná cena ušetrených peňazí z bežnej sumy
  7. Užívateľ zadá počet kusov produktu: **Data[“numberOfPieces“]**
  8. Užívateľ klikne na tlačidlo **Do košíku**

- Očakávaný výsledok
  - Je správne spočítaná celková suma
  - Sú správne načítané užívateľské údaje z prihláseného účtu

9. Užívateľ vyberie dopravu: **Data[“deliveryName“]**
10. Užívateľ vyberie platbu: **Data[“payment“]**
11. Užívateľ potvrdí súhlas s podmienkami
12. Užívateľ potvrdí súhlas s vekom, ak je povolený v administrácii
13. Užívateľ klikne na tlačidlo **Odeslat objednávku**
    - Očakávaný výsledok
      - Užívateľovi je zobrazená úspešná správa o vytvorení objednávky
14. Vykonáme test na filtrovanie vytvorenej objednávky v administrácií s id: 13
    - Očakávaný výsledok
      - Objednávka je nájdená v zozname objednávok

G Rozdelenie funkcií užívateľskej časti systému podľa priorít

- Priorita 1
  - Objednávanie produktov neregistrovaným zákazníkom v jednokrokovej objednávke
  - Objednávanie produktov neregistrovaným zákazníkom vo viackrokovej objednávke
  - Objednávanie produktov registrovaným zákazníkom v jednokrokovej objednávke
  - Objednávanie produktov registrovaným zákazníkom vo viackrokovej objednávke
  - Objemová zľava v jednokrokovej objednávke
  - Použitie zľavového kupónu pre neregistrovaného užívateľa v jednokrokovej objednávke
  - Použitie zľavového kupónu pre registrovaného užívateľa v jednokrokovej objednávke
  - Registrácia užívateľa do systému
  - Zobrazenie kategórií a podkategórií produktov
  - Zobrazenie variant produktov
  - Zobrazenie cien produktov
  - Google Tag Manager
  - Zboží.cz
  - Heureka
  - Facebook dynamický remarketing
  - Google Adwords remarketing
  - Sklik meranie konverzií
- Priorita 2
  - Prihlásenie užívateľa do užívateľského účtu
  - Odhlásenie užívateľa z užívateľského účtu
  - Zobrazenie SEO kategórie produktov
  - Zobrazenie SEO produktov
  - Množstvová zľava
  - Zľava a limit na produkty
  - Zobrazenie obrázkov u produktov
  - Objednávací košík
- Priorita 3
  - Články

  - Zobrazenie príznakov u produktov
  - Zobrazenie videa a odkazu u produktov
  - Zobrazenie súvisiacich a alternatívnych produktov
  - Hodnotenie produktov
  - Editácia osobných údajov v účte
  - Komentáre k objednávke
  - Zapamätanie si obsahu objednávacieho košíka
  - Odoslanie článku známemu
  - Zobrazenie kategórie článkov
- Priorita 4
  - Strážny pes
  - Posielanie noviniek emailom
  - Odporúčte nás

H Rozdelenie funkcií administračnej časti systému podľa priorít

- Priorita 1
  - Prihlásenie do administrácie
  - Tvorba nového produktu
  - Editácia objednávky
  - Nastavenie minimálnej ceny produktov
  - Vytvorenie dokladu k objednávke
  - Cena produktov voči cenníkom
  - Vytvorenie variant produktov zo šablóny
  - Vytvorenie novej kategórie
  - Vytvorenie novej podkategórie
  - Pridanie nového cenníku a overenie cien
  - Cenotvorba produktov a DPH
  - Platba na základe ceny objednávky
  - Platba na základe hmotnosti objednávky
  - Doprava na základe ceny objednávky
  - Doprava na základe hmotnosti objednávky
- Priorita 2
  - Tvorba objednávok pre registrovaného užívateľa
  - Tvorba objednávok pre neregistrovaného užívateľa
  - Tvorba objednávok pre anonymného užívateľa
  - Nastavenie výrobcov produktov
  - Pridanie obrázkov k produktu
  - Vytvorenie SEO pre produkty
  - Nastavenie katalógovej ceny produktov
  - Parametre produktov
  - Množstvová zľava produktov
  - Zľava a limit na produkt
  - Nastavenie SEO kategórie
  - Nezobrazovanie kategórie
  - Zľavy na kategórie a výrobcov

  - Doprava na základe cenníku zákazníka
  - Platba na základe cenníku zákazníka
  - Aplikácie 3. strán
  - Limity na produkt
  - Nastavenie a zmazanie objemovej zľavy
  - Generovanie zľavových kupóny
  - Menu pre kategóriu produktov
  - Dashboard
- Priorita 3
  - Vlastné stavy objednávok
  - Filtrovanie objednávky
  - Príznaky produktov
  - Video a odkazy k produktom
  - Dostupnosť produktov
  - Zrušenie používanie variant u produktov
  - Pridanie a odstránenie súvisiacich produktov
  - Zmazanie kategórie produktov
  - Duplikovanie kategórie
  - Hodnotenie produktov
  - Personalizácia
  - Vytvorenie článkov
  - Vytvorenie kategórie článkov
  - Tvorba horného menu
  - Menu pre kategóriu článkov
  - Menu - Rozdeľovník
  - Vyhľadávanie produktov
  - Zmazanie produktov
  - Importy a exporty
- Priorita 4
  - Odberatelia noviniek
  - Zber e-mailov - Pop up okna
  - Šablóny správ
  - Hromadné e-maily